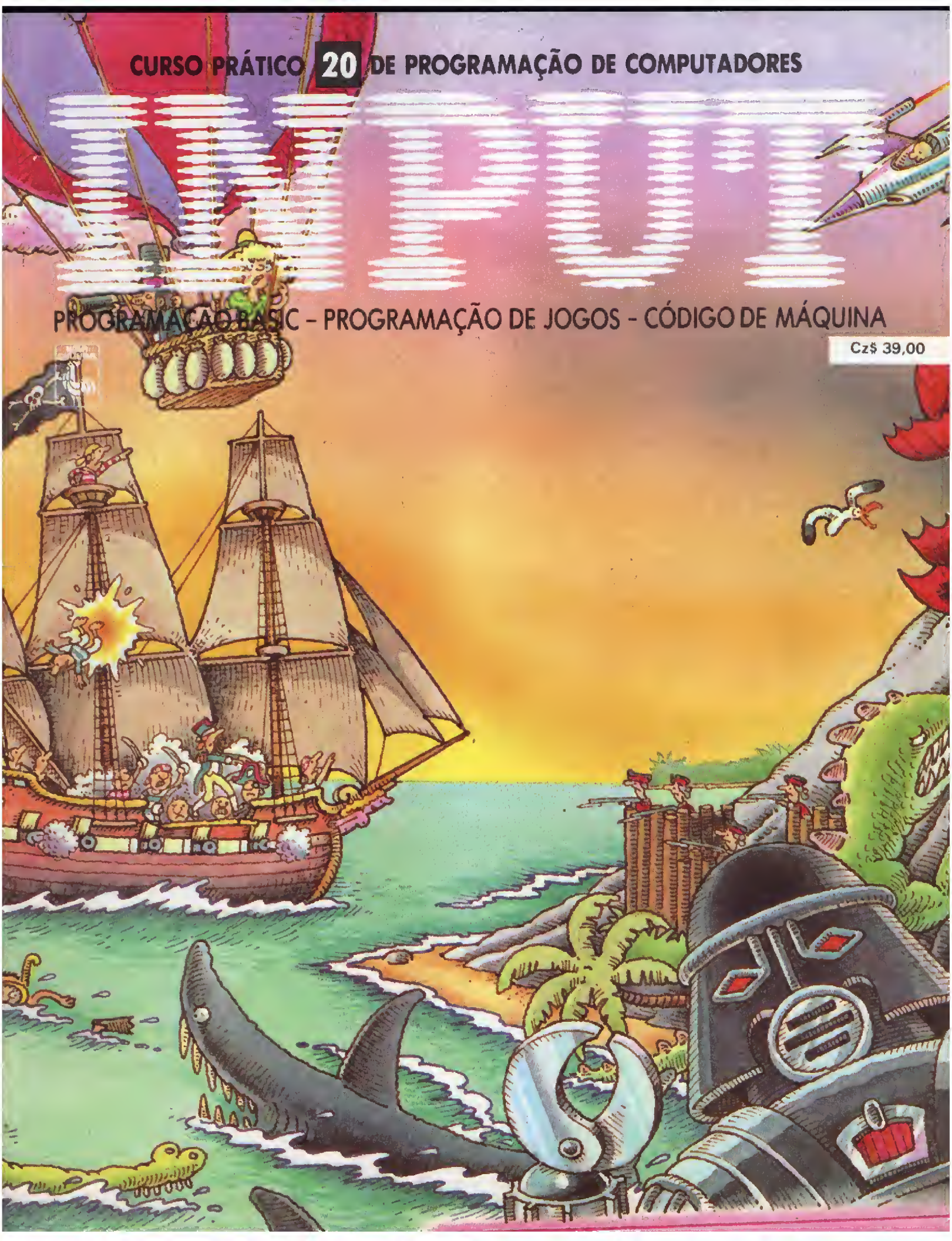
CURSO PRÁTICO 20 DE PROGRAMAÇÃO DE COMPUTADORES

FULL

PROGRAMAÇÃO BASIC – PROGRAMAÇÃO DE JOGOS – CÓDIGO DE MÁQUINA

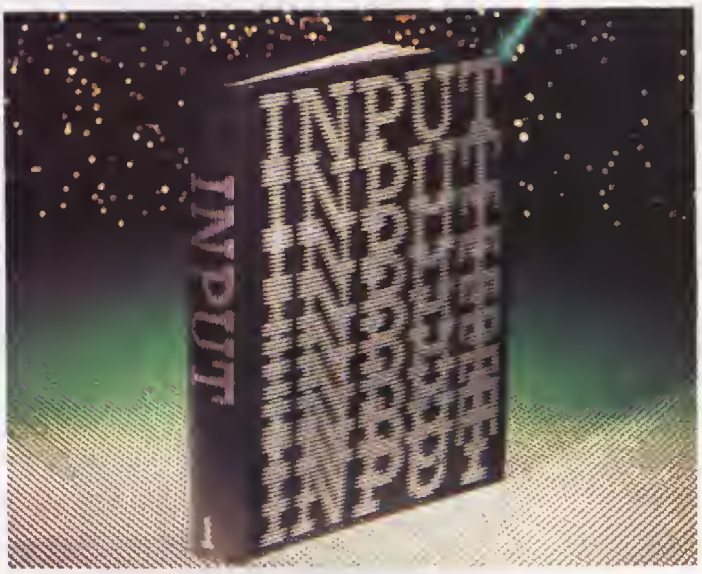
Cz$ 39,00



INPUT

Vol. 2 Nº 20

NESTE NÚMERO



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretor Editorial: Carmo Chagas

Editores Executivos: Antonio José Filho, Berta Sztark Amar

Editor Chefe: Paulo de Almeida

Editor de Texto: Cláudio A. V. Cavalcanti

Chefe de Arte: Carlos Luiz Batista

Assistentes de Arte: Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi

Secretária de Redação/ Coordenadora: Stefania Crema

Secretários de Redação: Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES

Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)

Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP

Tradução, adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira

Coordenação Geral: Rejane Felizatti Sabbatini

Editora de Texto: Ana Lúcia B. de Lucena

COMERCIAL

Diretor Comercial: Roberto Martins Silveira

Gerente Comercial: Joaquim Celestino da Silva

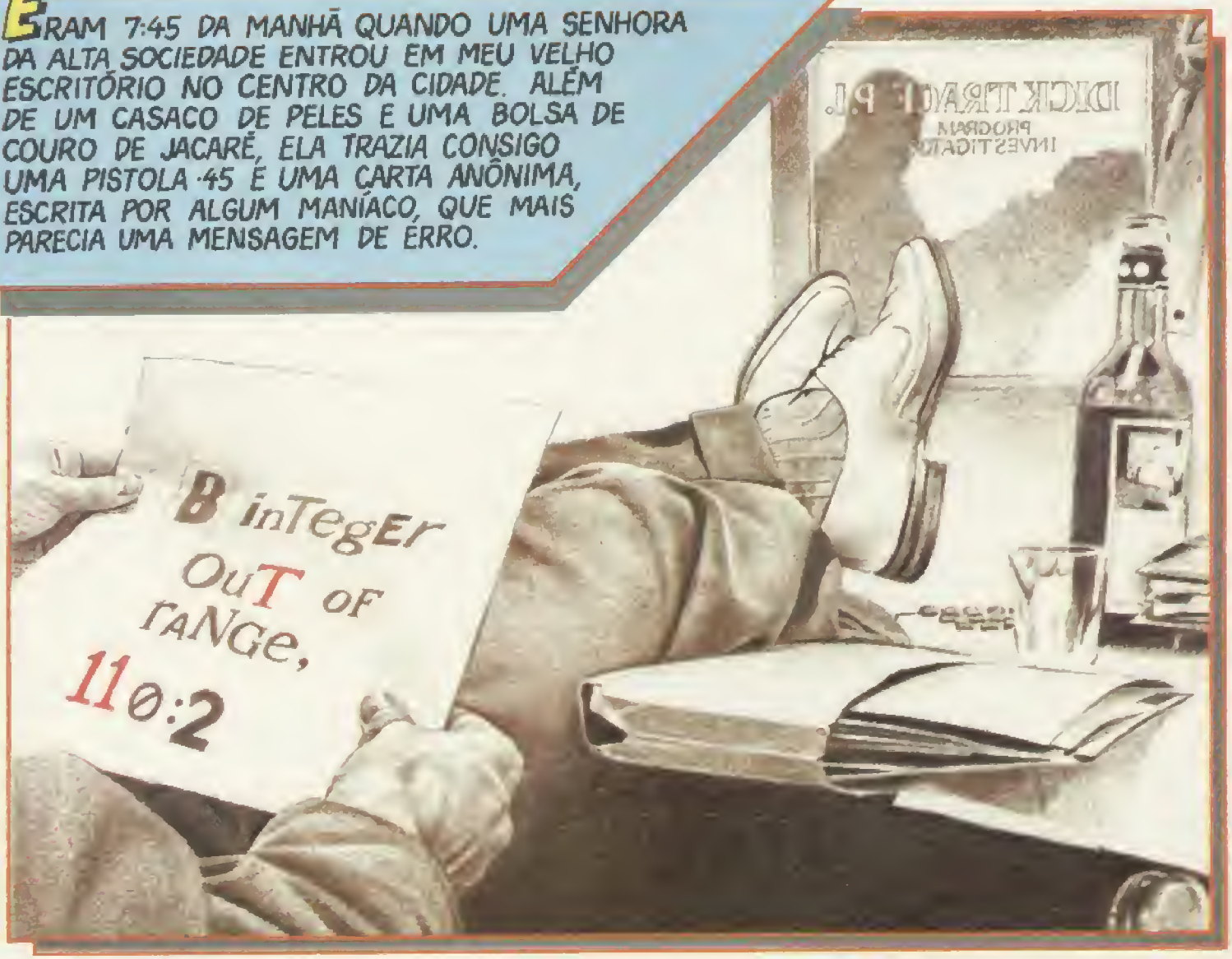
Gerente de Circulação: Denise Maria Mozol

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5988, de 14/12/1973).
Esta obra foi composta pela AM Produções Gráficas Ltda. e impressa pela Companhia Lithographica Ypiranga

RASTREAMENTO NO SPECTRUM

- O QUE FAZ UM PROGRAMA RASTREADOR
- A BUSCA DOS ERROS
- FUNCIONAMENTO E USOS DO PROGRAMA

ERAM 7:45 DA MANHÃ QUANDO UMA SENHORA DA ALTA SOCIEDADE ENTROU EM MEU VELHO ESCRITÓRIO NO CENTRO DA CIDADE. ALÉM DE UM CASACO DE PELES E UMA BOLSA DE COURO DE JACARÉ, ELA TRAZIA CONSIGO UMA PISTOLA .45 E UMA CARTA ANÔNIMA, ESCRITA POR ALGUM MANÍACO, QUE MAIS PARECIA UMA MENSAGEM DE ERRO.



Descubra onde se escondem aquelas linhas mal escritas. Interrogue o programa marginal que não funciona. Localize erro por erro com a ajuda do nosso programa rastreador.

É praticamente impossível digitar um programa longo — tal como o Assembler — sem cometer alguns erros. Mesmo que tenhamos conferido com todo o cuidado uma listagem, às vezes surgem erros que desafiam até programadores experientes, se não forem investigados com um recurso especial.

Para prosseguir no curso de linguagem de máquina, é essencial que seu Assembler funcione. Por isso, antes de mais nada, você deve encontrar todos os erros nele existentes. INPUT traz para os usuários do Spectrum de 48 K de memória um programa rastreador que será muito útil na busca dos erros eventualmente cometidos na digitação do Assembler. O TRS-Color, o TRS-80, o MSX e o Apple possuem programas rastreadores embutidos — são as funções **TRACE, TRACE ON** ou **TRON**.

O programa "TRACE" que apresentamos a seguir vem tanto em Assembly como em código hexadecimal. Assim, se seu Assembler não está funcionando, você pode usar o monitor da página 92 para entrar os códigos. Caso funcione bem, monte o programa TRACE com sua ajuda e grave-o em fita, para usá-lo em outros programas que contenham er-

ros. Se você não sabe se o Assembler está funcionando, use a montagem deste programa rastreador como um teste.

COMO USAR O PROGRAMA

Quando um programa BASIC tem um erro, seu micro geralmente informa em que linha está o problema. Isso pode ser suficiente para corrigir um erro em um programa simples e curto. Porém, quando se trata de um programa maior e mais complicado, a mensagem de erro muitas vezes é inútil. Uma linha sempre executada com sucesso pode, de repente, causar um erro porque em outro local do programa uma variável que ela usa assumiu um valor inadequado.

O programa rastreador aqui apresentado simplesmente escreve na tela o número da linha que está sendo executada. Ao contrário das instruções **TRACE** dos demais micros, nosso programa também escreve o código interno do comando BASIC usado no momento.

Para facilitar o uso do programa, convém ter à mão uma listagem do programa, seja ela sua ou de INPUT. Você poderá, assim, acompanhar o programa passo a passo, até atingir o ponto onde ocorreu o erro. Ficará mais fácil também descobrir os valores das variáveis e, ainda, se as condições dos **IF...THEN** estão sendo satisfeitas e se os **GOTO** estão indo para as linhas certas.

FUNCIONAMENTO DO TRACE

Um programa TRACE é um tipo muito especial de rotina em linguagem de máquina, pois funciona simultaneamente a outro programa — o seu, que está em BASIC. Normalmente, não se pode rodar dois programas ao mesmo tempo no computador. O TRACE, na verdade, não constitui uma exceção: ele não roda ao mesmo tempo que o programa em BASIC, mas utiliza um atributo do microprocessador que é a *interrupção*.

Rotinas que empregam esse artifício interrompem o programa principal a cada 20 milésimos de segundo — no Spectrum — e são executadas na fração de segundo que dura essa interrupção. Depois disso, o programa principal continua como se nada tivesse ocorrido, até a interrupção seguinte.

Programas BASIC são sempre interrompidos a cada 0,02 segundo enquanto estão funcionando, para que o computador verifique se alguma tecla foi pressionada. Rotinas que usam interrupções aproveitam-se desse padrão. A interrupção pode não ocorrer, tal como foi explicada, se houver algo conectado na expansão da memória.

Se uma linha BASIC for muito longa, ela poderá ser interrompida mais de uma vez durante sua execução. Nesse caso, o TRACE fornecerá o mesmo número de linha várias vezes. Por outro lado, se uma linha for muito curta — **PRINT** ou **RETURN** isolados, por exemplo — existe uma pequena chance de que TRACE não a detecte. Se isso acontecer, tente provocar uma pausa usando um **FOR...NEXT**.

S

O Spectrum tem dificuldade em imprimir na tela durante uma interrupção. Dois canais diferentes são usados para imprimir na parte superior da tela ou nas duas linhas inferiores, e, se mudarmos de canal enquanto o programa em BASIC está rodando, pode haver complicações.

Para contornar o problema, vamos colocar o número da linha diretamente na tela. Como cada caractere consome oito bytes, simplificamos a parte do programa que cuida disso, fazendo com que os números das linhas sejam sempre colocados no mesmo local da tela, fora da área que o BASIC costuma usar. Ao lado do número, coloca-se o código do comando BASIC que está sendo executado.

O programa TRACE não é chamado de dentro do BASIC, mas por meio de outra rotina em código que, por sua vez, é chamada pelo BASIC. É esta rotina que transfere o controle para o TRACE, durante as interrupções.

Não se esqueça de resguardar o topo da memória — os endereços a partir de 65109 devem estar livres e protegidos por **CLEAR 65109**. Se for usado o Assembler, as origens — ou endereços iniciais — estarão listadas. Se seu Assembler não funcionar, utilize o programa monitor da página 92. O endereço inicial é 65110.

Não se assuste se as traduções das linhas contendo saltos para rótulos que estão adiante da listagem, bem como das linhas que utilizam variáveis e bytes ainda não definidos, saírem erradas na tela. Elas são posteriormente corrigidas pelo próprio Assembler.

Não estranhe também se o Assembler parar a montagem antes de **end**, quando a parte inicial do programa, que contém as linhas **REM** com o programa-fonte, foi muito editada ou teve muitas linhas apagadas. Nesses casos, pode sobrar algum "lixo" no meio da listagem. Este não prejudica a interpretação pelo BASIC, mas confunde a do Assembler. Para contornar o problema, redigite a linha imediatamente posterior à última traduzida, e rode o programa novamente. Às vezes é necessário digitar várias linhas de novo.

```
org 65110
ld a,9                          3E 09
ld i,a                          ED 47
im 2                            ED 5E
ret                             C9
org 65120                    00 00 00
ld a,62                         3E 3E
ld i,a                          ED 47
im 1                            ED 56
ret                             C9
org 65129                       00 00
rst 56                          FF
push af                         F5
ld a,(23622)                 3A 46 5C
bit 7,a                         CB 7F
jr z,go                         28 02
pop af                          F1
ret                             C9
go di                           F3
push bc                         C5
push de                         D5
push hl                         E5
push ix                         DD E5
ld de,20726                  11 F6 50
ld (posn),de                 ED 53 FE FE
ld hl,(23621)                   2A 45 5C
call lineno                     CD B5 FE
ld de,20731                  11 FB 50
ld (posn),de                 ED 53 FE FE
ld hl,(23623)                   2A 47 5C
ld h,0                          26 00
call statno                     CD BB FE
ld hl,23286                     21 F6 5A
ld (hl),71                      36 47
ld de,23287                  11 F7 5A
ld bc,9                      01 09 00
ldir                            ED B0
keylp ld a,127                  3E 7F
in a,254                        DB FE
or 224                          F6 E0
cp 252                          FE FC
jr z,keylp                      28 F6
pop ix                          DD E1
pop hl                          E1
pop de                          D1
pop bc                          C1
pop af                          F1
ei                              FB
ret                             C9
lineno ld bc,-1000           01 18 FC
call prt                     CD CE FE
statno ld bc,-100            01 9C FF
call prt                     CD CE FE
ld bc,-10                    01 F6 FF
call prt                     CD CE FE
ld bc,-1                     01 FF FF
call prt                     CD CE FE
ret                             C9
prt xor a                       AF
prtlp add hl,bc                 09
inc a                           3C
jr c,prtlp                      38 FC
sbc hl,bc                       ED 42
dec a                           3D
add a,48                        C6 30
```

```
push hl                          E5
call print                 CD E8 FE
ld hl,posn                 21 FE FE
inc (hl)                         34
ld hl,(posn)               2A FE FE
call prtout                CD F5 FE
pop hl                           E1
ret                              C9
print ld bc,(23606) ED 4B 36 5C
ld h,0                           26 00
ld l,a                           6F
add hl,hl                        29
add hl,hl                        29
add hl,hl                        29
add hl,bc                        09
ex de,hl                         EB
ret                              C9
prtout ld b,8                    06 08
loop ld a,(de)                   1A
ld (hl),a                        77
inc h                            24
inc de                           13
djnz loop                        10 FA
ret                              C9
posn defw 0                      00 00
```

A listagem inclui três programas, e não apenas um. Cada um tem seu próprio endereço inicial. O primeiro, que começa em 65110, ativa o programa principal. O segundo, que começa em 65120, desliga o programa TRACE. O terceiro, que começa em 65129, faz o rastreamento.

As instruções **ld a, 9** e **ld i,a** colocam o número 9 dentro do registro I. Não há uma instrução que carregue o registro I diretamente com um número. O 9 é interpretado como o byte mais significativo de um vetor de interrupção que possui dois bytes. O byte menos significativo é fornecido pelo computador,

sendo geralmente igual a 255. Assim, o vetor de interrupção aponta para o endereço 9*256 + 255, que é igual a 2559. O valor contido nos endereços 2559 e 2560 é 65129, o endereço inicial do nosso programa TRACE.

O processo pode parecer confuso, mas rotinas com interrupções devem ser endereçadas indiretamente. Note que o endereço 2559 fica na ROM. Se colocássemos no registro I um número maior que 64, para apontarmos para um endereço da RAM, onde poderíamos colocar o número que quiséssemos — 65129 já estava lá —, veríamos que os caracteres se desmanchariam no vídeo.

O mnemônico **im 2** altera o modo de interrupção. A rotina que desliga o programa principal coloca o valor 62 no registro I; 62 era o conteúdo original desse registro. *im 1* restabelece o modo de interrupção normal.

O programa principal começa com a instrução **rts 56**. Ela determina que o microprocessador execute sua rotina normal de interrupção — fazendo a varredura do teclado e atualizando o relógio do sistema.

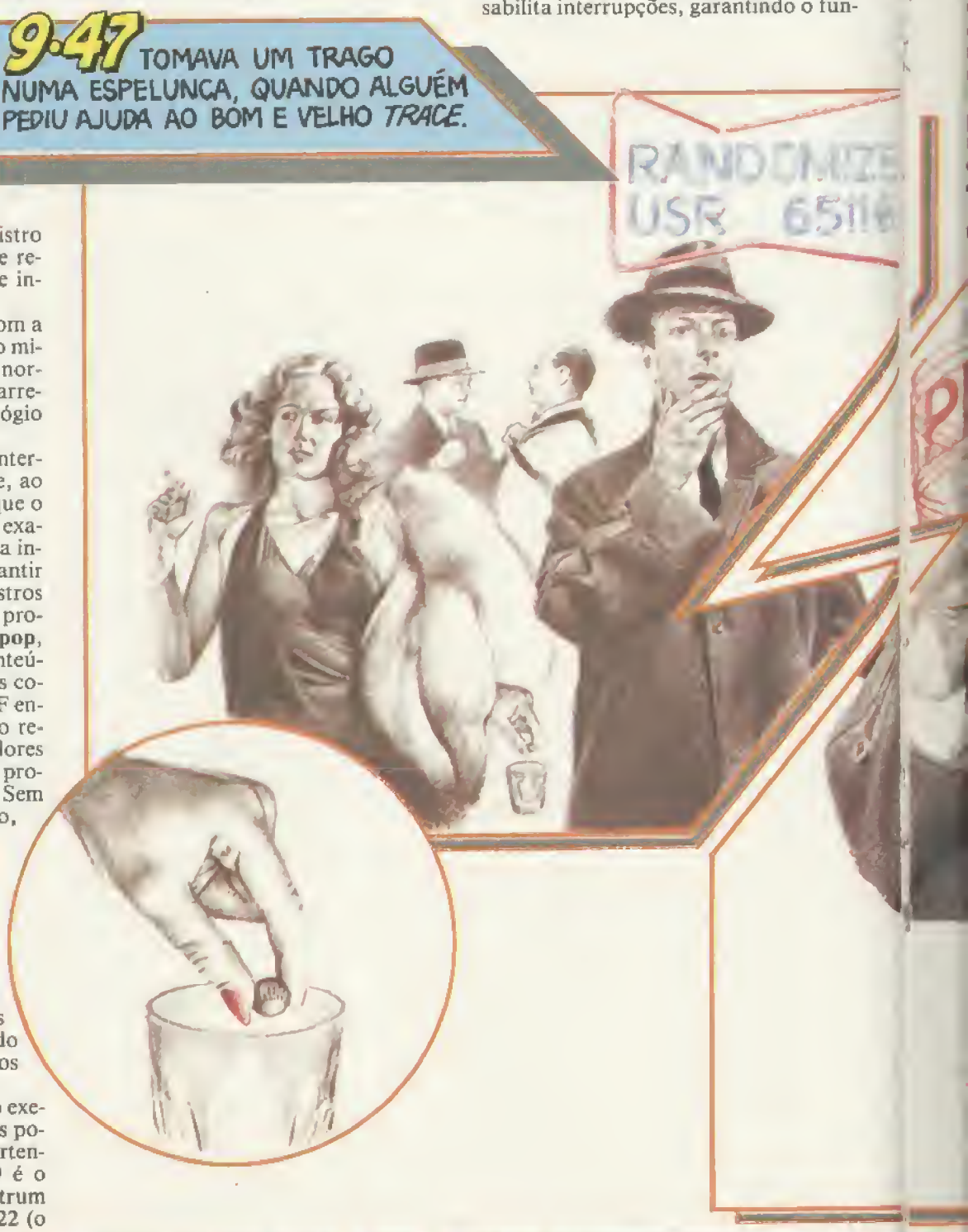
Ao programar uma rotina com interrupção, deve-se levar em conta que, ao final de sua execução, é essencial que o conteúdo de todos os registros seja exatamente igual ao que havia antes da interrupção. A única maneira de garantir isso é colocar o conteúdo dos registros na pilha, usando **push**, no início do programa, e retirá-lo de lá usando **pop**, quando o programa acabar. Os conteúdos de BC, DE, HL e IX são todos colocados na pilha. O conteúdo de AF entra antes porque o acumulador e o registro F — que contêm os sinalizadores — são utilizados para verificar se o programa BASIC está sendo rodado. Sem um programa BASIC funcionando, não há nada a ser rastreado.

Para verificar se existe um programa em curso, coloca-se o conteúdo da posição 23622 no acumulador. As posições 23621 e 23622 contêm o número da linha BASIC que está sendo executada. Embora os números das linhas BASIC sejam armazenados em formato alto-baixo na área do programa, aqui eles são armazenados no formato baixo-alto.

Se nenhum programa está sendo executado, o número contido nas duas posições será muito elevado para pertencer a uma linha BASIC — 9999 é o maior número de linha que o Spectrum aceita. Assim, o conteúdo de 23622 (o byte mais significativo) é colocado no acumulador e a instrução **bit 7,a** examina o bit mais significativo do conteúdo de A. Se esse bit for 1, o programa não está sendo executado. Uma instrução **bit** verifica o valor de determinado bit de uma posição ou registro. Por exemplo, **bit 4,a** testa o quarto bit do acumulador.

Se o bit mais significativo do byte mais significativo da linha BASIC que está sendo executada for 0, o sinalizador zero é estabelecido e **jr z,go** faz com que o programa vá para a próxima ocorrência do rótulo **go**. Se o bit 7 do acumulador for 1, o sinalizador não é estabelecido e o saldo não ocorre. O programa continua na próxima instrução, que recupera o valor de AF da pilha e retorna ao interpretador BASIC.

Uma vez que haja um programa sendo rodado e que ele tenha sido transferido para o rótulo **go**, a instrução **di** desabilita interrupções, garantindo o fun-



cionamento contínuo de nossa rotina de interrupção.

A instrução **ld de,20726** coloca no registro DE o endereço da posição da tela imediatamente anterior àquela onde colocaremos um caractere. Como o programa usará muito esse endereço, ele é colocado na variável **posn**, por meio do comando **ld posn,de**. Nosso Assembler interpreta a variável **posn** como uma posição de memória que tem um nome. Como nenhuma instrução permite colocar um número diretamente dentro de uma variável ou posição de memória, usamos um registro como intermediário.

Coloca-se no par HL o conteúdo das posições 23621 e 23622, que é o número da linha BASIC que está sendo executada no momento. Podemos então chamar a sub-rotina de rótulo **lineno**.

O número da linha BASIC em HL está em hex. Para convertê-lo para decimal, começamos por calcular o algarismo dos milhares: subtraímos 1000 do número em HL repetidamente, contando quantas vezes foi possível fazer tal operação. Colocamos, assim, o valor −1000 no par BC e chamamos a sub-rotina **prt**, que faz as subtrações. Nessa sub-rotina, a primeira coisa que o microprocessador faz é **xor a** — isto é, um ou exclusivo. Em linguagem de máquina, um **xor** sempre age no registro A; assim, **xor a** faz um ou exclusivo do acumulador com ele mesmo, o que sempre resulta em 0. Esta é uma maneira rápida de colocar 0 no acumulador, já que **ld a,0** tem um byte a mais.

add hl,bc subtrai 1000 do conteúdo de HL. Somar −1000 é mais rápido que subtrair 1000, já que dispensa cuidar do sinalizador *carry* ("vai um"). O acumulador é, então, incrementado e atua como contador. **jr c,prtlp** verifica o sinalizador carry e salta quando ele é estabelecido.

Se considerarmos o conteúdo de BC entenderemos como tudo funciona. Vimos no artigo da página 142 que os números negativos são representados dentro do computador por números positivos muito grandes, mas de comprimento limitado. O par de registros BC contém dezesseis números 1 binários, menos 1000 em decimal. Se qualquer número maior que 1000 for somado a BC, o valor do resultado ultrapassará a capacidade do par HL e o sinalizador carry será estabelecido.

Quando isso ocorre, há um salto, 1000 é novamente subtraído de HL, o contador é incrementado e o processo recomeça. Ele se repete até que o conteúdo de HL seja menor que 1000; a adição com BC, então, não estabelece o sinalizador carry. Neste ponto, porém, a subtração já foi feita e a incrementação do contador ocorreu uma vez mais que o necessário. Assim, somamos 1000 a HL — na realidade subtraímos −1000 — e decrementamos o acumulador.

O acumulador contém, agora, o algarismo dos milhares. Adicionando 48, obtemos o código ASCII do caractere. O conteúdo de HL é reservado na pilha para quando formos calcular as centenas, dezenas e unidades. Em seguida, chamamos a sub-rotina **print**.

A primeira instrução dessa sub-rotina é **ld bc, (23606)**, que coloca o conteúdo das posições 23606 e 23607 em BC. Essas posições contêm a variável do sistema que aponta para o endereço inicial do conjunto de caracteres da ROM. O conteúdo de H é reduzido a zero e o do acumulador é transferido para L, que conterá, assim, o código ASCII do caractere que queremos escrever. O registro HL é usado, então, como um acumulador de 16 bits.

Para colocar na tela o caractere desejado, temos que obter seu formato dentro do conjunto de caracteres que fica na ROM. Como cada caractere tem oito bytes, o endereço inicial dos oito bytes que fornecem o padrão do caractere é oito vezes o código ASCII do caractere, mais o endereço inicial do conjunto de caracteres.

Em vez de multiplicar o código ASCII por oito, é mais fácil duplicá-lo — somando-o consigo mesmo — três vezes. Observe que a operação não é feita no registro A porque o valor certamente ultrapassará oito bits — 48 X B = 384, posição inicial do caractere 0, é maior que 255, máxima capacidade de um registro de oito bits. Essa conta, porém, não ultrapassará os dezesseis bits.

Para obter o endereço do primeiro byte do caractere desejado, adiciona-se o resultado da operação ao conteúdo de BC. Os conteúdos de HL e DE são intercambiados para que HL possa ser usado novamente.

ret determina o retorno do microprocessador da sub-rotina e **ld hl, posn** coloca a posição da tela contida em **posn** no registro HL. A instrução indireta **inc (hl)** incrementa o conteúdo de **posn**. A sub-rotina que coloca o caractere na tela — **prtout** — é, então, chamada.

A primeira instrução dessa sub-rotina é **ld b, 8**, que coloca 8 no registro B. Ele será usado como contador para os oito bytes do caractere que serão transferidos para a tela. O acumulador é carregado com o conteúdo da memória cujo endereço está em DE, ou seja, com o primeiro byte do caractere. O conteúdo do acumulador é transferido para a posição de memória cujo endereço está em HL (o local apropriado da tela). Exceto em um comando de ação em bloco, não se pode transferir o conteúdo de uma posição de memória diretamente para outra sem utilizar um registro como intermediário. Não há uma instrução do tipo **ld (hl), (de)**, por exemplo. Essa transferência requer, portanto, duas instruções, e o que ela realmente faz é colocar a primeira linha de oito pontos do caractere na tela.

O conteúdo do registro H é, então, incrementado. Como H contém o byte mais significativo do par HL, o conteúdo desse par aumenta em 256, de forma a corresponder ao endereço da próxima linha do caractere.

Incrementando o conteúdo de DE, obtemos o endereço da próxima linha do caractere, dentro do conjunto de caracteres, e **djnz loop** faz o microprocessador voltar à sub-rotina **prtout**, para colocar na tela os outros bytes do caractere. Este laço é repetido oito vezes, incrementando H a cada volta, para colocar na tela o próximo byte abaixo do anterior, e incrementando DE, para obter o padrão dos oitos pontos a serem colocados na tela, do conjunto de caracteres da ROM. Ao mesmo tempo, a instrução **djnz** (decrementa saltando se não for zero) decrementa o registro B. Assim, quando o processo estiver na oitava repetição — e os oito bytes que compõem o caractere tiverem sido colocados na tela —, o caractere correspondente ao algarismo dos milhares estará no vídeo, o registro B conterá zero, a condição não-zero do comando **djnz** não será satisfeita e o microprocessador passará à próxima instrução, que é **ret**, retornando para onde foi chamada a sub-rotina.

O comando **pop hl** recupera os dois últimos bytes colocados na pilha. Se olharmos para trás, veremos que se trata do resto que ficou em HL após o cálculo do algarismo dos milhares. **ret** manda o microprocessador de volta à linha onde está **statno ld bc, −100** e o processo se repete para calcular o algarismo das centenas. Quando este for colocado na tela ao lado do algarismo dos milhares, o algarismo das dezenas — e, depois, o das unidades — é calculado da mesma maneira e colocado na tela na posição adequada.

Depois disso, o microprocessador retorna para onde a sub-rotina **lineno** foi chamada. O valor 23731 é colocado em **posn**, um endereço da tela um pouco à direita de onde foi colocado o número da linha. Obtém-se, assim, espaço suficiente para quatro dígitos do número e um espaço em branco.

O conteúdo das posições 23623 e 23624 é colocado em HL; estas posições contêm o número da instrução BASIC que está sendo executada no momento. O valor máximo que o código de uma instrução BASIC pode assumir é 128; o byte mais significativo é, portanto, desnecessário e deve ser reduzido a zero pela instrução **ld hl, 0**. A sub-rotina que cuida da conversão do número para decimal, bem como de sua impressão na tela, é executada novamente, só que desta vez é chamada por meio do rótulo **statno**, já que são necessários apenas três dígitos.

Quando o código da instrução BASIC é novamente impresso na tela, o microprocessador volta ao programa principal, onde executa uma pequena sub-rotina que coloca no vídeo os caracteres invertidos. Como a rotina leva menos de 20 milésimos de segundo, não há a menor chance de percebermos que um caractere foi desenhado normalmente e depois invertido.

ld hl,23286 é o endereço da posição anterior ao primeiro dígito na tabela de atributos, que fornece uma "moldura" ao número. O número 71 — que produz fundo preto e caracteres brancos — é colocado nesta posição. O endereço da próxima posição é colocado em DE e 9, no par BC, que será usado como contador.

A instrução de armazenamento em bloco **ldir** coloca o conteúdo da posição de memória dada por HL na posição dada por DE, decrementa B e verifica se o valor de B foi reduzido a zero. Se não foi, o processo é repetido. Em outras palavras, a instrução copia os atributos fundo preto e caracteres brancos do primeiro caractere nos outros nove caracteres adjacentes.

PAUSA NO PROGRAMA

As nove instruções seguintes permitem parar temporariamente o programa TRACE — e, também, o programa BASIC — a qualquer momento. A instrução **in** recebe informações vindas de uma das portas de entrada. No nosso caso, estamos interessados no teclado, que envia dados ao microprocessador pela porta 254. Queremos verificar se alguma tecla da porção inferior direita do teclado — de B até **BREAK/SPACE** — foi pressionada. Assim, **ld a,127** coloca 127 dentro do acumulador para ser usado como parâmetro da instrução **in**. Juntos, os comandos **ld a,127** e **in a, 254** colocam o valor enviado pela porção citada do teclado no acumulador.

De B a **BREAK/SPACE** existem apenas cinco teclas, cada qual representa-

da por um bit dentro do número enviado ao microprocessador. Sobram, portanto, três bits. Se executarmos a operação lógica **or** entre o conteúdo do acumulador e 224, faremos com que os três bits mais significativos se tornem 1 — 224 é 11100000 em binário.

Quando uma tecla não está sendo pressionada, seu bit dentro do valor enviado é 1. Quando ela é pressionada, este valor muda para zero. Para provocar uma pausa no TRACE, deve-se pressionar duas teclas simultaneamente: **<SIMBOL SHIFT>** e **<BREAK/SPACE>**. Com a pausa, o teclado pode ser usado normalmente para editar linhas, mesmo que o programa esteja "ligado".

Quando pressionamos **<BREAK/SPACE>**, o bit zero do número enviado passa de 1 para 0. **<SIMBOL SHIFT>** faz o mesmo ao bit um. Se não pressionarmos nenhuma das duas teclas e cada um dos três bits mais altos for transformado em um, o número enviado pela porção do teclado em questão será 255, ou 11111111. Mas, se pressionarmos as duas teclas ao mesmo tempo, o número enviado será 252, ou 11111100.

Assim, quando o número enviado pela porta 254 entra no acumulador, é comparado com 252 pela instrução **cp 252**. Se o acumulador contiver 252, o sinalizador zero será estabelecido. Então, **jr z** (salto relativo se zero) faz com que a rotina seja repetida. Enquanto as duas teclas permanecerem pressionadas, as repetições prosseguirão. Como o nosso programa interrompeu o programa principal, este também não prosseguirá enquanto o microprocessador não terminar a execução da rotina.

Se nenhuma tecla é pressionada, o processador passa às instruções **pop ix**, **pop hl**, **pop de**, **pop bc** e **pop af**, que recuperam o conteúdo original dos registros anteriores à interrupção.

A instrução **ei** habilita a ocorrência de interrupções e **ret** faz o microprocessador voltar ao interpretador BASIC, que continuará a execução do programa principal.

A última instrução da listagem Assembly, **defw posn**, não terá códigos correspondentes no programa-objeto que o Assembler vai criar. Esse tipo de instrução é usado pelo Assembler apenas para reservar dois bytes, onde serão colocados números utilizados pelo programa. No nosso caso, o que fica armazenado ali é o valor de **posn**.

A instrução **defw** (definir dois bytes) reserva um espaço para colocar dados, permitindo também que se dê um nome a ele — **posn**, por exemplo. Outra instrução semelhante é **defb**, que "define um byte".

COMO ENCONTRAR OS ERROS

Para usar o TRACE, carregue o programa BASIC a ser corrigido. Em seguida, reserve o topo da memória com **CLEAR 65109** e carregue o TRACE.

Antes de rodar o programa rastreador, grave-o em fita. Se o seu Assembler estiver funcionando, faça-o pela via usual. Assim, você gravará o programa-fonte, das linhas **REM**, junto com o resto do Assembler. Quando o programa rastreador estiver funcionando, é melhor gravá-lo em fita, já montado em linguagem de máquina. Para isso, digite:

```
SAVE "TRACE" CODE 65110,176
```

Para carregá-lo, digite primeiro **CLEAR 65109** e, em seguida:

```
LOAD "" CODE
```

Se o Assembler não estiver funcionando, use a opção de gravação do seu monitor. Para ligar o TRACE, digite:

```
RANDOM USR 65110
```

Nada acontecerá até que um programa em BASIC seja executado, com **RUN**. Depois disso, aparecerão no vídeo o número da linha e o código do comando que estiver sendo executado.

O problema mais fácil de se detectar com o auxílio do programa rastreador é o "laço sem fim". Se você observar que os mesmos números de linha se repetem mais e mais vezes, pode estar certo de que há algo errado em algum **GO-TO**. Por outro lado, se certas linhas nunca forem executadas, é provável que algum **IF...THEN** contenha condições erradas. Caso queira observar os eventos mais detalhadamente, use a pausa.

Para desligar o programa rastreador, digite:

```
RANDOM USR 65120
```

ARTE GRÁFICA EM SEU MICRO

Conhecendo mais a fundo os comandos gráficos de seu micro, você poderá fazer melhor uso dos recursos disponíveis. E mais: saberá contornar todas as suas limitações.

Já reunimos muita informação a respeito dos comandos gráficos. Ainda assim, é possível que não estejamos fazendo o melhor uso deles.

Os comandos de cor, por exemplo, são muito mais versáteis do que parecem à primeira vista e fazem muito mais do que simplesmente pintar blocos. Aqui estão algumas idéias e técnicas que talvez o ajudem a aprimorar seus gráficos.

Os comandos **PLOT** e **DRAW** do Spectrum podem ser usados de várias maneiras, mas sabemos que nem sempre produzem o efeito desejado. Muitas vezes, isto se deve às limitações da tela de alta resolução gráfica, que não aceita cores diferentes em pontos adjacentes dentro de um mesmo quadrado na tela de texto. Mas há outros fatores que interferem nos resultados. O melhor a fazer é tentar aproveitar ao máximo o efeito obtido, seja ele qual for.

Por exemplo, se o método de sombreamento utilizado apresenta um resultado pouco uniforme, elabore o programa de modo que o efeito pareça proposital. E se algumas cores estão se sobrepondo em certas áreas do trabalho, ou se você precisa de mais de duas cores num mesmo quadrado, projete o desenho de modo que qualquer mudança de cor aconteça num novo quadrado.

Restarão ainda os problemas com as curvas mas, com um certo esforço, será possível diminuir seus efeitos.

O importante é que, diante da impossibilidade de obter um determinado efeito, você pelo menos esteja preparado para adaptar e usar todos os recursos disponíveis. Daí, sim, na medida em que aprendermos mais sobre a máquina, procuraremos desenvolver novos métodos de refinamento do desenho.

COMO SOMBREAR A TELA

Sabemos que o Spectrum dispõe de eficientes recursos para a elaboração de desenhos (veja artigos das páginas 113 e 232, por exemplo).

Mostraremos agora como sofisticar ainda mais os gráficos, explorando melhor os comandos de desenho e de cor que já conhecemos. E por que não aumentar nosso repertório de figuras ou aprender a utilizá-las como base para outras?

Este pequeno programa desenha pontos em posições aleatórias. Observe quanto tempo leva para preencher a tela toda.

```
10 LET x=INT (RND*256)
20 LET y=INT (RND*176)
30 PLOT x,y
40 GOTO 10
```

Como você viu, demora bastante, o que torna o método pouco aconselhável para quem quer preencher completamente uma grande área num gráfico. Entretanto, o programa é muito útil quando se pretende sombrear uma área — e isto ele faz com certa rapidez.

Com o simples acréscimo de uma linha criamos um efeito ainda melhor. Adicione a seguinte linha a seu programa e rode-o novamente:

```
25 IF (X>35 AND X<90) AND (Y>
   35 AND Y<90) THEN GOTO 10
```

Logo se observa que o computador vai deixando um quadrado limpo de pontos — ou seja, vazio. A linha 25 verifica se os valores de x e y estão entre 35 e 90; o computador salta, em seguida, para a linha 10, para gerar novos valores. Obtemos, então, uma área vazia entre 35 e 90, em ambas as direções; forma-se, assim, um quadrado nessas coordenadas.

Poderíamos utilizar essa técnica num jogo: por exemplo, depois de escrever algo num quadrado, o resto da tela seria preenchido gradualmente, enquanto as palavras continuariam visíveis. A mesma técnica também será útil para destacar uma mensagem na tela, como o título de um programa ou jogo.

- ADAPTAÇÃO DOS RECURSOS DISPONÍVEIS
- USE MELHOR AS CORES
- COMO SOMBREAR E COLORIR A TELA DO SPECTRUM
- MAIS SOBRE O CIRCLE E O PAINT NO MSX
- PSET, PRESET E COLOR NO TRS-COLOR
- O FLASH NO APPLE

As condições na linha 25 parecem ser complicadas. No artigo da página 34, vimos como usar **AND** e **OR** para incluir mais de uma condição num **IF...THEN**. O significado dessas funções coincide exatamente com a tradução dos dois termos — ou seja, E e OU, respectivamente. Na linha 25, portanto, x tem que ser maior que 35 **AND** (E) menor que 90, **AND** y tem que ser menor que 90 **AND** maior que 35; só assim o computador executará o **GOTO** 10.

Os parênteses entre as duas metades da linha separam as condições para x e y (que na verdade são as mesmas, mas devem ser separadas por um **AND**). No nosso exemplo, poderíamos retirar os parênteses, pois utilizamos somente **AND**. Caso usássemos tanto **AND** quanto **OR**, os parênteses seriam essenciais. Veremos o que o **OR** faz no exemplo seguinte. Podemos mudar a área que pretendemos deixar intacta da maneira que quisermos — experimente, por exemplo, substituir o **AND** entre parênteses por um **OR**. Em vez de quadrado vazio, obteremos uma cruz.

Mude a linha 25 para:

```
25 IF (X>110 AND X<145 AND Y>40
   AND Y<136) OR (X>40 AND X<
   215 AND Y>70 AND Y<100) THEN
   GOTO 10
```

Você pode deixar qualquer área vazia, mas não se esqueça de que, quanto mais complicada for esta área, mais condições terá o **IF...THEN**, e mais tempo o computador levará para verificá-las. Mesmo um triângulo — ou um círculo — requer muitas condições e, assim, o computador demorará para desenhar os pontos. Aconselhamos, no caso de sombreamento, restringir as áreas a linhas horizontais e verticais, pois elas levam menos tempo para serem verificadas e, portanto, aparecem logo na tela.

DESENHO E COR

Em artigos anteriores vimos como desenhar no Spectrum figuras bastante detalhadas. Os principiantes em programação gráfica, porém, deparam-se com um problema: de um modo geral, essa máquina apresenta uma grande limitação quando se trata de desenhar uma figura e sombreá-la com cor. Caso não saiba do que estamos falando, veja este exemplo:

```
 5 BORDER 7: PAPER 7: INK 4:
 CLS
10 FOR x=1 TO 10
20 PRINT " ■■■■■■■■■■ "
30 NEXT x
40 PLOT  INK 2;0,175
50 DRAW  INK 2;120,-80
60 INK 1: CIRCLE 100,95,50
```

O programa preenche uma área em verde, usando blocos gráficos, depois traça uma linha diagonal vermelha e um círculo azul — pelo menos, deveria ser assim. Na prática, porém, todos os quadrados por onde a linha passa mudam de cor. É claro que existem maneiras de se evitar que isso ocorra, mas a maioria delas precisa de longos programas para que o Spectrum manipule as informações.

Mais adiante, em outro artigo, veremos como fazê-lo. Por enquanto, ficaremos com uma alternativa simples: evitar as áreas problemáticas. Foi o que fizemos nos programas de gráficos anteriores, como o do campo de golfe, apresentado na página 233. Nesse caso, escolhemos primeiro a cor da tela e, depois, desenhamos as linhas, observando que nenhuma delas passasse sobre áreas já ocupadas por caracteres gráficos.

Para que você entenda melhor e, ao mesmo tempo, experimente um trabalho com arcos, digite o programa a seguir. Ele desenha um carro colorido.

```
 80 BORDER 2: PAPER 6: INK 0:
 CLS
 90 FOR n=8 TO 15
100 CIRCLE 80,47,n: CIRCLE 180
,47,n
110 NEXT n
120 PLOT 62,51: DRAW 38,-5,-PI
130 DRAW 60,0: DRAW 40,0,-PI
200 FOR n=1 TO 10: READ a,b,c:
DRAW a,b,c: NEXT n
210 PLOT 112,48: DRAW 45,0:
DRAW 22,20,-PI/2
220 DRAW 7,15: DRAW -35,23:
DRAW -40,0: DRAW 0,-58
230 PLOT 115,104: DRAW 34,0:
DRAW 30,-19
240 DRAW -64,-5,-.25: DRAW 0,
24
250 PLOT 40,83: DRAW 52,5,.3:
DRAW 6,20,.3
260 PLOT 30,55: DRAW -5,1:
DRAW 0,5: DRAW 5,1
270 PLOT 240,56: DRAW 5,1:
DRAW 0,5: DRAW -5,1
280 PLOT 36,75: DRAW  INK 2;
184,0
290 PRINT  OVER 1; INK 2;AT 13
,4;"■";AT 13,28;"■"
300 FOR n=31 TO 0 STEP -1:
PLOT 0,n: DRAW  INK 4;255,0:
NEXT n
500 DATA 40,8,.2,0,10,0,-30,15
,.2,-20,5,.2
510 DATA -40,25,0,-60,-2,.1,
-50,-25,.2
520 DATA -10,-20,-.25,0,-8,0,
31,-3,.2
```

As linhas 90 a 110 desenham uma série de círculos, cada um maior que o anterior, formando as rodas do carro. O Spectrum não consegue traçar círculos perfeitos numa área quadrada. Em con-

seqüência das diferenças de curvatura, alguns pontos se perdem, ou melhor, não são incluídos em curva alguma. Como esses pontos não são desenhados, o pneu do carro adquire uma aparência pontilhada. Poderíamos encarar tal efeito como um problema, mas não há razão para isso, já que, na verdade, a roda ganhou um visual ainda mais realístico. Como afirmamos, para se desenhar bem é necessário fazer bom uso das características da máquina, e não se atrapalhar com elas.

A linha 120 desenha um ponto na coordenada 62,51, que é a posição inicial das linhas que formam o carro. Usando o laço **FOR...NEXT** na linha 200, e as linhas 120 e 130, o Spectrum lê (**READ**) as informações necessárias para desenhar (**DRAW**) o contorno do carro. Os dados (**DATA**) para o laço **FOR...NEXT** estão nas linhas 500 a 520. Note que todas as curvas estão na forma **x, y, z**, que é o comando para desenhar uma linha curva, ou seja, um arco, cuja curvatura é especificada por **z**.

Os primeiros dois números são os mesmos de sempre: definem o quanto acima e o quanto à direita do último ponto queremos o próximo ponto. O terceiro número especifica o ângulo da curva. O comando que desenha o capô curvo do carro é **DRAW 38,−5−PI**. Tente mudar o último número (PI é mais ou menos 3,14) e veja a diferença de curvatura do capô.

Os detalhes internos ao contorno constituem seções do programa: a porta e seu vidro estão nas linhas 210 a 240; o vidro traseiro, na linha 250; os pára-choques, nas linhas 260 a 270 e a lista na linha 280. As linhas 290 e 300 simplesmente dão um toque colorido ao desenho: a linha 290 coloca as lanternas (vermelhas) e a linha 300 faz a grama sob o carro.

Posicionando cuidadosamente o carro e a grama, evitaremos o problema de ter duas cores num mesmo quadrado. Observe, no programa acima, a grama sob o carro. Este foi posicionado com suas rodas chegando até o fundo de um quadrado, de modo que o ponto seguinte (onde começa a grama) fica em outro quadrado, não interferindo, assim, na cor dos pneus.

Como se vê, é necessário planejar em detalhes a posição de cada elemento. Mas nem sempre isso é possível: no nosso caso, por exemplo, a listra vermelha altera a cor de algumas partes do carro. O importante, porém, é evitar alterações de cor nos lugares onde elas apareceriam muito.

Para praticar com o **DRAW**, aconselhamos que tente mudar algumas partes do carro; somente assim será possível saber exatamente o que cada linha faz e, ao mesmo tempo, familiarizar-se com o comando. O ideal seria podermos imaginar a figura acabada sem precisar desenhá-la desde o começo.

O USO DO FLASH

O **FLASH**, um comando de utilização muito simples, é ideal para destacar mensagens na tela, principalmente em jogos. Ele não usa parâmetro nenhum e, em geral, vem antes do comando **PRINT**. Não se esqueça, porém, de que o **FLASH** é um comando separado do **PRINT**: por essa razão, deve vir numa outra linha ou separado por dois pontos. Exemplo:

```
FLASH:PRINT" PISCA-PISCA "
```

Quando o computador encontra o comando **FLASH** em uma linha de programa, ele faz "piscar" tudo o que for mandado para a tela a partir daquela linha, ou seja, o conteúdo de todos os **PRINT** que vierem depois de **FLASH**. O **FLASH** mostra alternadamente o inverso e o normal dos caracteres exibidos na tela.

Digite e rode o seguinte programa:

```
10  HOME : VTAB 10
20  FLASH
30  PRINT "ISTO ESTA PISCANDO"
40  FOR T = 0 TO 2000: NEXT
50  HOME : VTAB 10
60  PRINT "   ISTO  TAMBEM   "
70  FOR T = 0 TO 2000: NEXT
80  GOTO 10
```

Tecle <CTRL-C> para parar e liste o programa. Tudo continua piscando, pois o **FLASH** ainda está ativado. Para desativá-lo, digite **NORMAL** e aperte a tecla <**RETURN**>. Liste novamente.

O comando **NORMAL** desativa o **FLASH** e pode ser usado tanto dentro de programas como fora deles ou, em outras palavras, tanto no modo direto como no indireto. Substitua as linhas 60, 70 e 80 e adicione a linha 90. As modificações são as seguintes:

```
60  NORMAL
70  PRINT "ISTO NAO ESTA"
80  FOR T = 0 TO 2000: NEXT
90  GOTO 10
```

Se apertarmos <CTRL-C> no "ISTO NÃO ESTÁ" e depois listarmos o programa, veremos que a listagem não pisca, pois interrompemos o programa quando este estava em modo **NORMAL**. Mas, se apertarmos <CTRL-C> no "ISTO ESTÁ PISCANDO", a listagem também piscará, pois interrompemos o programa no modo **FLASH**, ou melhor, com o **FLASH** ativado.

O comando **INVERSE** mostra os caracteres em modo inverso, ou seja, letras pretas num fundo branco.

O programa que se segue faz a apresentação dos três comandos em seus respectivos modos.

```
10  HOME : VTAB 10
20  FLASH
30  PRINT : PRINT "  FLASH  "
40  GOSUB 110
50  INVERSE
60  PRINT : PRINT " INVERSE "
70  GOSUB 110
80  NORMAL
90  PRINT : PRINT "  NORMAL"
100  END
110  FOR T = 0 TO 2000: NEXT
120  RETURN
```

O programa que apresentamos a seguir emprega gráficos já vistos e poderia ser usado para ensinar inglês a crianças.

O programa desenha figuras para as três primeiras letras do alfabeto. Para a letra A, desenha uma "apple" (maçã); para a letra B, uma "ball" (bola) e, para a C, um "cat" (gato). A ampliação do programa, ou seja, o desenho de figuras para as outras letras, ficará a cargo de cada um. A figura relacionada à letra D poderia ser, por exemplo, um "dog" (cão) e estaria situada a partir da linha 4000. Para dizer ao computador aonde ir caso a letra escolhida seja a D, basta acrescentar uma vírgula e 4000, depois do 3000 na linha 60. A figura da letra E poderia ser colocada a partir da linha 5000 e assim por diante.

Se quiser aperfeiçoar o programa, escreva também o nome da figura e a letra escolhida, como mostram as ilustrações da bola e do gato. Para isso, utilize a sub-rotina que desenha letras em tela de alta resolução, já apresentada mais

de uma vez em INPUT (por exemplo, no artigo da página 354). Deixamos a adaptação desta sub-rotina como um desafio a programadores mais experientes.

```
5 SCREEN 0:COLOR 1,15,15
10 PI=4*ATN(1):CLS:LOCATE 9,10
20 PRINT"ME DE UMA LETRA => ?"
30 A$=INKEY$:IFA$<"A"ORA$>"Z"TH
EN30
40 LOCATE 28,10:PRINTA$
50 FORT=1TO500:NEXT
60 A=ASC(A$)-64:ON A GOTO 1000,
2000,3000
70 GOTO 30
1000 SCREEN2:COLOR15,11,12:CLS
1010 CIRCLE(128,96),40,6,,,1
1020 PAINT(128,96),6
1030 FOR Z=1 TO 50 STEP .3
1040 PSET((100+Z*.5)+RND(1)*25,
80+RND(1)*40),15
1050 NEXT
1060 CIRCLE(90,70),40,12,0,PI/2
,1
1070 CIRCLE(112,50),10,12,,,.5
1080 CIRCLE(140,55),10,12,,,.5
1090 PAINT(112,50),12
1100 PAINT(140,55),12
1110 GOTO 3280
2000 SCREEN2:COLOR15,1,9:CLS
2010 C1=INT(RND(1)*14)+1
2020 C2=INT(RND(1)*14)+1
2030 CIRCLE(128,96),10,C1,,,.7
2040 CIRCLE(128,96),50,C1,,,.7
2050 PAINT(140,90),C1
2060 CIRCLE(128,96),30,C2,,,.7
2070 CIRCLE(128,96),10,C2,,,.7
2080 PAINT(140,90),C2
2090 GOTO 3280
3000 SCREEN2:COLOR1,3,4:CLS
3010 REM CORPO/CABEÇA/OLHOS
3020 CIRCLE(128,106),40,1,,,1.5
3030 CIRCLE(128,46),20,1,,,1.15
3040 CIRCLE(121,40),5,1,,,.5
3050 CIRCLE(135,40),5,1,,,.5
3060 PAINT(128,46),1
3070 PAINT(128,106),1
3080 REM ORELHAS
3090 X1=108:X2=113:X3=103
3100 LINE(X1,15)-(X2,28),9
3110 LINE(X1+40,15)-(X2+40,28),
9
3120 LINE(X2,28)-(X1,41),9
3130 LINE(X2+40,28)-(X1+40,41),
9
3140 LINE(X1,41)-(X3,28),9
3150 LINE(X1+40,41)-(X3+40,28),
9
3160 LINE(X3,28)-(X1,15),9
3170 LINE(X3+40,28)-(X1+40,15),
9
3180 PAINT(105,28),9:PAINT(150,
28),9
3190 REM CAUDA
3200 CIRCLE(75,106),35,1,1.2*PI
,1.8*PI,1
3210 CIRCLE(75,106),33,15,1.2*P
I,1.8*PI,1
3220 CIRCLE(128,50),1,15,,,1
3230 REM BIGODE
3240 LINE(128,52)-(150,55),15
3250 LINE(128,52)-(150,53),15
3260 LINE(128,52)-(106,55),15
3270 LINE(128,52)-(106,53),15
3280 FOR T=0 TO 2000:NEXT:GOTO 5
```

Logo no início, o programa pede por uma letra. A linha 60 é responsável pelo cálculo do valor da letra, que será guardado na variável **A**, e também pela escolha da sub-rotina relacionada à letra escolhida. No nosso programa, se a letra escolhida foi A, B ou C o programa saltará para a linha 1000, 2000 e 3000, respectivamente. Na linha 60, não há opção para as letras maiores que C; para estas letras, o programa volta para a linha 30.

A sub-rotina que desenha a maçã começa na linha 1000. As linhas 1030 a 1050 desenham pontos brancos em posições aleatórias, para dar um visual mais realista ao desenho. A linha 1060 desenha um arco que vai de 0 a PI/2 graus, representando o pequeno caule.

A bola começa na linha 2000; C1 e C2 são cores aleatórias de números 1 a 15. Suponhamos que a bola esteja dividida em duas: a bola maior, que será pintada na cor C1, e a bola menor (dentro da maior), que receberá a cor C2. Pintamos a bola maior primeiro porque, no MSX, a cor com que colorimos a figura deve ser a mesma do seu contorno. Seria impossível, por exemplo, fazer em branco o círculo que divide as duas bolas e, depois, pintar a maior de azul e a menor de verde. Nenhuma cor utilizada para colorir aceitará outra cor como delimitação de contorno. Nossa saída foi desenhar na cor C1 a bola maior e pintá-la nessa cor, antes de desenhar e, também, pintar na cor **C2** a bola menor. Observe que foi preciso redesenhar o círculo menor na cor C2 (linha 2070) para que a bola menor o aceitasse como parte dela. Omita a linha 2070 para ver o que acontece.

A sub-rotina que desenha o gato começa na linha 3000. As linhas 3110, 3130, 3150 e 3170 desenham a orelha esquerda, que está 40 pontos à direita da outra — por isso somamos 40 às mesmas coordenadas x da outra orelha. As linhas que desenham a cauda — 3200 e 3210 — seguem o mesmo princípio do desenho do caule, na linha 1060. A linha 3220 desenha o nariz.

Já vimos como usar o **PSET** no desenho de círculos e curvas seno e cosseno. Vamos agora explorar um pouco mais seu funcionamento e o de seus relacionados **PRESET**, **PCLS** e **COLOR**.

O COMANDO PSET

Podemos dizer que o **PSET** "acende", na cor escolhida, a menor unida-

de gráfica em qualquer **PMODE**.

No **PMODE 4**, somente um ponto é aceso; nos **PMODE** 2 e 3, acendem-se dois pontos ao mesmo tempo, e nos **PMODE** 0 e 1, quatro. Não confunda o **PSET** empregado dessa maneira com o **PSET** usado anteriormente no comando **LINE** (veja o artigo da página 113).

Rode este programa:

```
10 PMODE 0,1
20 PCLS
30 SCREEN 1,1
40 FOR X=0 TO 255
50 Z=(X-127)/10
60 Y=95-150*Z/(1+Z*Z)
70 IF Y<0 OR Y>191 THEN 90
80 PSET (X,Y,5)
90 NEXT
100 GOTO 100
```

O programa desenha um gráfico no modo de duas cores, mas poderia também desenhá-lo em **PMODE** 4 ou **PMODE** 2. Este é o gráfico de uma função matemática obscura e foi escolhido porque produz um traço interessante.

Ao usar **PSET** devemos dizer à máquina onde queremos acender o ponto (ou conjunto de pontos) e em que cor. As cores disponíveis dependem do **PMODE** utilizado ou do conjunto de cores escolhido.

A cor da tela será a que tiver menor número, entre as cores disponíveis no conjunto escolhido, a não ser que queiramos mudá-la.

Como veremos mais tarde, quando usarmos **PSET** no modo de duas cores precisaremos especificar a cor de maior número; caso contrário, não veremos nada. No modo de quatro cores, a situação é diferente, já que podemos escolher qualquer uma das três cores de maior número.

Retornando ao programa: o conjunto de cores preto e amarelo foi escolhido na linha 30. As linhas 40, 50 e 60 calculam valores para x e y e a linha 80 acende o ponto na coordenada x, y. O último argumento é a cor que o ponto receberá — no nosso caso, 5 (amarelo).

MODO DE QUATRO CORES

Tente mudar a linha 10 para:

```
10 PMODE1,1
```

Agora, rode o programa novamente. Não se preocupe se nada acontecer — é isto o que se espera. Na verdade, estamos acendendo pontos em amarelo num fundo também amarelo. Temos que mudar o último argumento do **PSET**, na linha 80, para selecionar uma cor diferente. Neste conjunto de cores, podemos escolher entre ciano (6), magenta (7) e laranja (8). Tente mudar o argumento 5, na linha 80, para 6, 7 ou 8. Se quiser, mude o conjunto de cores na linha 30 para:

```
30 SCREEN1,0
```

Agora, podemos usar cores de números 1 a 4, embora 1 acenda o ponto na mesma cor do fundo.

Só como curiosidade: se escolhermos o conjunto de cores 0 e usarmos 5, 6, 7 ou 8 como último argumento do **PSET**, a máquina não enviará mensagem de erro, pois automaticamente subtrai 4 dos números.

O COMANDO PRESET

O **PRESET** é o inverso do **PSET**, ou seja, ele "apaga" o ponto especificado nos argumentos entre parênteses. Pode-se também imaginar o **PRESET** como acendendo um ponto, ou conjunto de pontos, na mesma cor do fundo.

Para observar o funcionamento do **PRESET**, rode o programa novamente e, então, mude a linha 80 para:

```
80 PRESET(X,Y)
```

Mas preste atenção: não use o comando **RUN** para rodar o programa com a linha alterada, pois se o fizer a tela será limpa.

Para verificar como o **PRESET** apaga ponto por ponto, digite **GOTO** 30 e, em seguida, tecle <**RETURN**>.

A COR DA TELA

É possível mudar a cor da tela para qualquer uma das cores do conjunto escolhido. Para isso, basta acrescentar o número da cor junto ao **PCLS**, na linha 20.

Supondo que a linha 30 seja **SCREEN1,1**, tente as seguintes mudanças na linha 20: **PCLS6**, **PCLS7** e **PCLS8**. **PCLS5** tem o mesmo efeito de **PCLS**, pois ambos limpam a tela na cor amarela. Terminadas as mudanças, faça a linha 20 **PCLS** novamente.

FUNDO E PRIMEIRO PLANO

Como vimos no artigo da página 113, o comando **LINE** usa **PSET** e **PRESET** de uma maneira diferente da que acabamos de ver. Quando empregado com o comando **LINE**, o **PSET** diz ao computador para desenhar na cor do primeiro plano e o **PRESET** diz que o desenho deve ser feito na cor do fundo.

Quando ligamos o TRS-Color, a cor do primeiro plano é a de maior número do conjunto de cores (veja seu manual para os números das cores) e a cor de fundo é a de menor número.

Mas suponha que estamos num modo de quatro cores e queremos traçar uma linha em uma das outras cores do conjunto. Não teremos problema, pois existe um comando em BASIC que nos permite escolher as cores do fundo e do primeiro plano.

Digite este programa para ver como funciona o comando **COLOR**:

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 FOR K=1 TO 4
50 FOR J=1 TO 4
60 COLOR K,J
70 LINE(0,50*K+10*J-55)-(255,50
*K+10*J-55),PSET
80 LINE (0,50*K+10*J-52)-(255,5
0*K+10*J-52),PRESET
90 NEXT J,K
100 GOTO 100
```

O programa desenha pares de linhas paralelas, uma na cor de fundo e outra na cor de plano. Não se pode ver todas as linhas, pois o fundo ou o plano ajustados em verde coincidem com a cor da tela.

O comando **COLOR** na linha 60 produz a variação das cores. Observe que os comandos **LINE** não são alterados durante o programa, exceto nas coordenadas finais.

O primeiro número depois do comando **COLOR** é a cor do plano e o segundo, a cor do fundo. Nada impede que a cor do fundo seja a mesma do plano, embora não haja muita utilidade nisso.

Se usarmos somente **PCLS** no programa, as cores de fundo e tela serão iguais, mas, se especificarmos um número junto ao **PCLS**, as cores podem ser diferentes. Da mesma maneira, a cor do plano nem sempre é a mesma cor com que desenhamos. O único comando gráfico que faz uso das cores de plano e fundo é o **LINE** e, como vimos, ainda assim podemos desenhar tanto na cor de fundo como na de plano.

CRIE SUA PRÓPRIA AVENTURA

Agora que você já viu como programar uma aventura, é hora de partir para criações originais. Eis como usar o jogo de INPUT como base para elaborar o seu próprio programa.

A esta altura, você deve ter um jogo completo de aventura gravado em fita. Durante seu desenvolvimento, vimos como reunir no programa os elementos que o compõem. Agora, você aprenderá a utilizar o jogo já pronto como base para elaborar suas próprias aventuras.

Algumas dicas de como proceder para fazer seu jogo foram dadas ao longo dos últimos artigos. Todos os detalhes serão tratados aqui em maior profundidade.

Nem sempre poderemos ser muito específicos sobre as alterações necessárias, uma vez que elas dependerão muito das características da sua aventura; mas você poderá fazê-las com facilidade, se seguir as instruções. Algumas técnicas parecerão confusas no início, mas, começando com uma aventura simples e curta, você logo se sentirá à vontade para programar jogos mais complexos. Não tente fazer muitas alterações de uma só vez; vá devagar, estudando as seções deste artigo e fazendo as alterações uma a uma.

Comandos BASIC que não foram bem compreendidos podem se tornar mais claros com uma consulta à seção de *programação BASIC* correspondente — um grande número de comandos já foi abordado por essa seção de INPUT.

Os usuários do TK-2000 e do Spectrum de 16k não poderão estender muito sua aventura, o que não significa que não possam aumentá-la um pouco ou acrescentar outros locais. Sempre é possível, por exemplo, trocar alguma característica da aventura por um novo local; tudo dependerá do que se considerar mais importante.

A ESCOLHA DO TEMA

Antes de escrever sua aventura, será preciso escolher uma trama ou história.

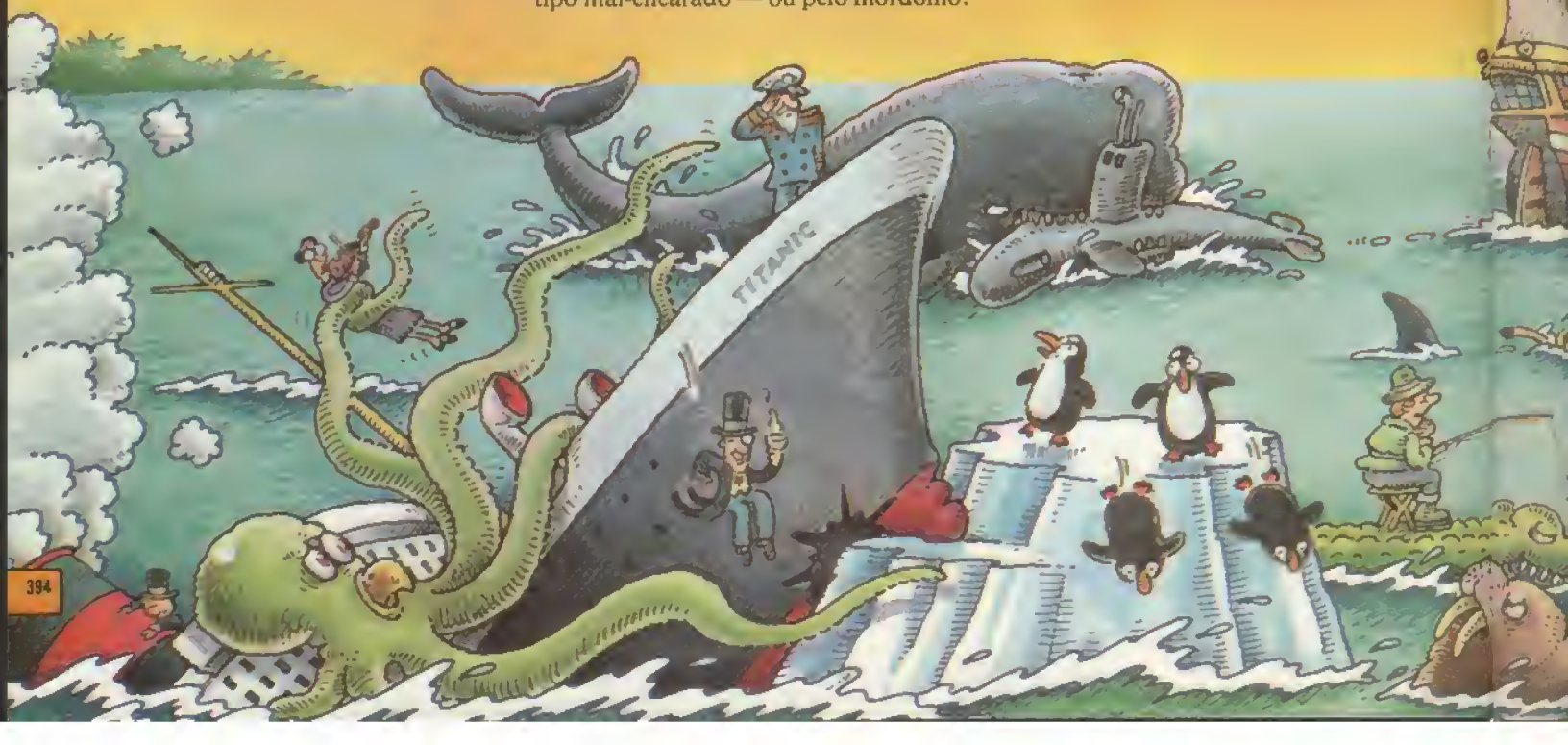
A estrutura de uma aventura bem-sucedida em geral não foge a certos padrões — há começo, meio e fim, ordenados conforme a seqüência em que os problemas devem ser resolvidos. Mas não é necessário ser Agatha Christie para programar aventuras. Existem muitas fontes de idéias, como você já deve ter percebido; em todo caso, aqui vão algumas sugestões.

Podemos basear nossa aventura em um assassinato. O jogo começaria, por exemplo, numa sala, onde um corpo esfaqueado jaz sem vida sobre o tapete ensangüentado. O objetivo do aventureiro seria descobrir o assassino. Substituiríamos, então, o fiscal da Receita por um tipo mal-encarado — ou pelo mordomo. Ele poderia ajudar ou atrapalhar o jogador com pistas verdadeiras ou falsas.

Se você quiser manter o tema da "caça ao tesouro", encontrará várias maneiras de utilizá-lo. Recorra, por exemplo, ao clichê tradicional, com piratas e tudo, ou transforme o jogador em sobrevivente de um desastre aéreo que vitimou todos os membros de uma expedição. Mandar o jogador rumo ao futuro, por outro lado, pode ser uma boa idéia. Procurando novos mundos, ele acaba preso em um planeta hostil, anos-luz distante da civilização, por causa de um defeito em sua nave espacial. O jogador teria como objetivo achar um jeito de voltar à Terra. Os locais poderiam ser cheios de perigo e haveria bastante espaço para incluir — ou ocultar — diversas rotas de fuga.

Esta idéia de fuga sugere outros temas de aventuras, em que o objetivo do jogador seria, por exemplo, escapar de lugares como Alcatraz, San Quentin ou o Presídio da Ilha Grande. Use livros e jornais como fontes de inspiração. Talvez você até encontre um mapa do lugar real para fazer o mapa da aventura!

Casos de espionagem também podem dar aventuras interessantes, assim como fatos históricos. As cruzadas, por exemplo, podem



se revelar um cenário muito interessante, assim como qualquer outra guerra ou batalha.

E, como sugestão final, um tema da moda: o aventureiro sobrevive ao holocausto nuclear. Os elementos para o jogo são quase infinitos: mutantes, busca de proteção contra a radiatividade, ameaça de gangues de bandidos famintos, procura de água e comida não contaminadas e assim por diante.

NOVOS LOCAIS

A aventura de INPUT é bem pequena; as suas logo serão bem maiores que ela.

Se seguirmos as instruções das páginas 226 a 231, teremos um mapa que poderá ser introduzido no computador. O mapa da nossa aventura tem 24 locais (6x4) e apenas 12 deles são utilizados. Podemos aproveitar o programa original — o que dá menos trabalho, mas exige mais habilidade — ou escrever um novo — o que envolve mais esforço, porém menos confusão. Se você escolher a primeira alternativa, deverá carregar o programa original da fita ou listá-lo através da impressora. As modificações dependerão do tama-

nho do novo mapa. Se ele for menor ou igual ao antigo, conserve a numeração original. Caso contrário, será preciso fazer um novo desenho, numerando-o segundo os mesmos princípios.

Pronto o mapa, você deverá fazer novas descrições dos locais. Elas vão substituir as antigas, que ficavam a partir da linha 5000, no programa.

Após cada descrição, informe as possíveis saídas do local, usando as variáveis **N**, **S**, **E** e **W**, que correspondem a norte, sul, leste e oeste. Elas podem conter os valores 1 e 0. 0 significa que não há saída naquela direção, enquanto 1 quer dizer o contrário. O esforço de digitar linhas **REM** indicando os locais pode valer a pena.

Em seguida, modifique as linhas 330 a 350 que contêm **ON...GOSUB**. O primeiro número que se segue ao **GOSUB**, na linha 330, se refere à linha onde o computador vai encontrar a descrição do local 1. Se não houver local 1 na sua aventura (não é preciso incluir todas as posições do mapa), 0 é usado. O próximo número se refere à descrição do local 2, e assim por diante. Cada posição do mapa deve ter seu número.

MOVIMENTO

Se o tamanho do mapa for diferente do original, você precisará alterar as linhas 1000 a 1040 da rotina de movimento. Mais especificamente, as linhas que se referem às direções norte e sul — 1010 e 1030 — devem ser modificadas se o mapa não tiver seis posições de largura. Isto porque, para seguir nestas direções, subtraímos ou adicionamos 6 ao número do local. A nova largura do mapa deve substituir o 6.

OBJETOS

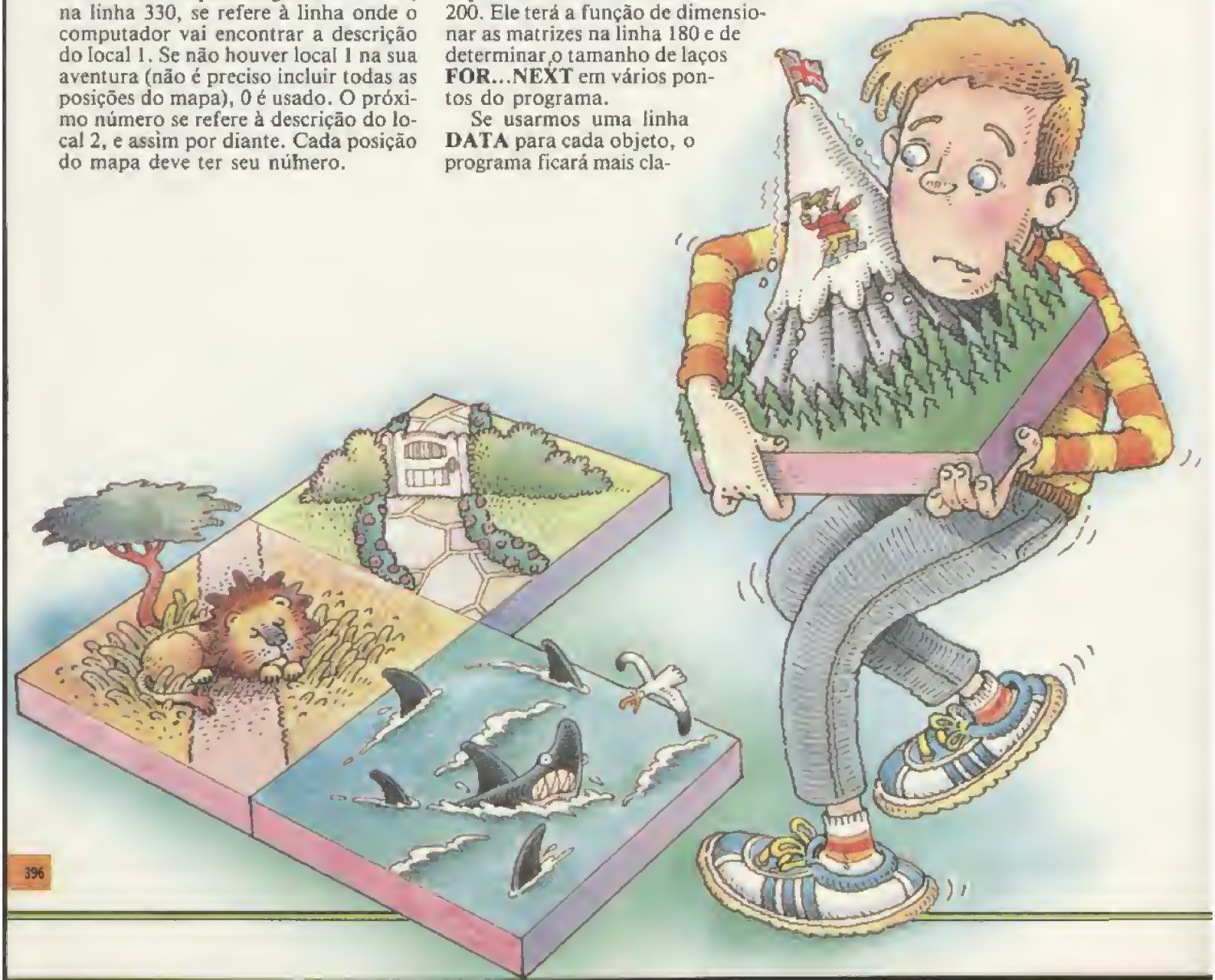
Grandes modificações deverão ser feitas nas linhas 160 a 260, pois os objetos certamente serão diferentes.

O número de objetos da nova aventura dá o valor da variável **NB**; deve ser o primeiro elemento na linha **DATA** 200. Ele terá a função de dimensionar as matrizes na linha 180 e de determinar o tamanho de laços **FOR...NEXT** em vários pontos do programa.

Se usarmos uma linha **DATA** para cada objeto, o programa ficará mais claro. No entanto, se o número de objetos for muito grande, pode-se optar por colocar mais de um por linha. Em todo caso, a ordem deve ser precisa, pois cada grupo de três dados contém elementos de três diferentes matrizes. A ordem é: número do local onde se encontra o objeto, descrição curta e descrição longa. Se o objeto só aparece mais tarde na aventura — após ser encontrado, por exemplo, como o olho — ou se surge ao acaso, como o fiscal da Receita, o número do local deve ser zero.

NOVAS PALAVRAS

É preciso preparar uma lista de todas as palavras permitidas ao jogador. Ela deve incluir comandos simples, como "AJUDAR" e "LISTAR", bem como comandos com duas palavras, tais como "PEGAR LÂMPADA" e "MA-



TAR SENHORIO".

Os comandos de duas palavras dividem-se em **V$** e **N$** — verbo e substantivo, respectivamente —, embora nem sempre de acordo com a definição gramatical. Vamos considerar todos os comandos simples e todas as primeiras palavras dos comandos compostos como sendo verbos. Eles devem ser agrupados de acordo com seu significado — "MASTIGAR" e "COMER", ou "CHEIRAR" e "FAREJAR", por exemplo, ficarão juntos. Cada um desses grupos terá um número — anote-os, pois será difícil decorar a que grupo de palavras correspondem.

Agora vamos alterar o programa. A rotina que trata dos verbos vai da linha 110 a 150. Os verbos devem ser colocados — cada qual seguido de seu número — nas linhas **DATA** 140 e 150.

Não se esqueça de redimensionar as matrizes, na linha 130, de acordo com o número total de verbos utilizados.

ROTINAS QUE ENTENDEM OS VERBOS

Cada categoria — ou número — de verbo vai precisar de uma rotina diferente.

É difícil dar instruções específicas a respeito dessas rotinas, porque um verbo usado em uma aventura pode não servir para nada em outra.

Contudo, algumas rotinas — como "PEGAR" e "LARGAR" — podem ser úteis em qualquer aventura. Você não precisará modificá-las, a menos que o roteiro de seu jogo seja muito inovador. Da mesma forma, não será necessário alterar a rotina "LISTAR", nas linhas 1070 a 1130, caso os nomes das matrizes e a variável **NB** (número de objetos) permaneçam os mesmos.

Uma rotina que sempre permite adaptações é a da lâmpada, já que situações que envolvem acendê-la são muito comuns em aventuras.

As rotinas restantes, de um modo geral, não podem ser aproveitadas. Você precisará, assim, escrever novas rotinas. Ao fazê-lo, lembre-se de que uma das finalidades é verificar se o jogador escolheu corretamente o objeto e o local para executar aquilo que o verbo indica. Se o local não é adequado, o programa deve orientá-lo com mensagens do tipo "AQUI NÃO". Seja qual for o resultado da ordem dada, o jogador precisa ser informado — isto é, qualquer palavra que entre no computador deve provocar uma mensagem.

Depois de planejar as rotinas de execução das ordens verbais, coloque-as no programa. Se seu programa tem uma numeração parecida com a da aventura de INPUT, o lugar dessas rotinas é entre as linhas 1070 e 2999.

O computador selecionará a rotina correspondente ao verbo usado pelo jogador. A linha 510 é a responsável por isso, e deverá ser modificada. Para tanto, consulte os números de sua lista de verbos. Coloque, então, na frente do **ON...GOTO**, o endereço inicial das rotinas, na mesma ordem numérica da lista dos verbos correspondentes.

ROTINA DE AUXÍLIO

A última rotina a ser modificada é a que trata de "AJUDAR" o jogador. Verifique onde esse auxílio é necessário e use **IF...THEN** para dar uma mãozinha ao aventureiro.

Outras características da aventura original devem ser modificadas ou eliminadas, conforme as exigências do novo jogo. É o caso, por exemplo, da rotina que faz o coletor de impostos aparecer. Há também a alternativa de começar o jogo em uma posição diferente; para isso, basta alterar a linha 280.

LISTA DE VARIÁVEIS

Para que você entenda melhor o funcionamento do programa original, fizemos uma lista das variáveis e matrizes por ele utilizadas.

R$() matriz contendo os verbos e respostas do jogador.
R() matriz de números de verbos.

Elementos correspondentes das duas matrizes acima referem-se ao mesmo verbo.

OB() matriz com o número do local onde se encontra o objeto.
OB$() matriz com a descrição curta do objeto.
SI$() matriz com a descrição longa do objeto.

Elementos correspondentes das três matrizes acima referem-se ao mesmo objeto.

NB número de objetos da aventura, usado para dimensionar matrizes e tamanhos de laços **FOR...NEXT**.
L local onde o jogador se encontra no momento.
LA sinalizador do estado da lâmpada; contém 1, se a lâmpada está acesa, e 0, se está apagada.
TA sinalizador do coletor de impostos.
N,S,E,W saídas de um local; contém 1, se há saída naquela direção, e 0, se não há.
I$ entrada ou resposta completa do jogador, antes de ser dividida em verbo e substantivo.
V$** verbo em **I$.
N$** substantivo em **I$.
I contém um número correspondente a um verbo; usado para selecionar a rotina de execução adequada.
IN número de objetos que estão sendo carregados pelo jogador.
A$ resposta à pergunta "Quer jogar novamente?"
G contém o número do objeto deixado de lado pelo jogador — que é o elemento **G** na matriz **OB**.

SPECTRUM

O funcionamento do programa do Spectrum é um pouco diferente, devido à ausência de comandos **ON...GOSUB** e **ON...GOTO** nesta máquina. Não há desvantagem nisso. O programa funciona muito bem e sua modificação não apresenta maiores dificuldades.

DESCRIÇÕES DOS LOCAIS

O primeiro passo para adaptar o programa do Spectrum é incluir as descrições dos locais de seu novo mapa.

Elas são colocadas no final do programa, da mesma forma que na aventura original. Devem permanecer em ordem e seguidas da linha que contém as saídas — variáveis **N, S, E** e **W**, correspondendo a norte, sul, leste e oeste, acompanhadas dos números 1 ou 0, indicando se há ou não saída naquela direção. Não se esqueça das linhas **REM**, para saber onde está cada descrição na listagem.

NOVAS PALAVRAS

Faça uma lista de todas as palavras que o jogador poderá usar durante o jogo. Neste momento, interessam particularmente os verbos — a primeira palavra em um par, ou palavras usadas isoladamente e que nem sempre são verbos no sentido gramatical. Reúna as palavras que têm o mesmo sentido (e, portanto, o mesmo efeito no desenrolar da aventura), como "PEGAR" e "APANHAR" ou "MATAR" e "ATIRAR". Cada grupo precisará de um número. Inclua-os também na lista.

Os verbos seguidos de seus números devem ser colocados nas linhas **DATA** 140 e 150. Não se esqueça de deixá-los entre aspas.

Deveremos ainda redimensionar as matrizes na linha 120 e ajustar o tamanho do laço **FOR...NEXT**, na linha 130, de acordo com o número de palavras utilizadas.

ROTINAS QUE ENTENDEM OS VERBOS

Cada categoria de verbos precisará de uma rotina.

É difícil dar instruções específicas sobre elas, uma vez que quase sempre servem apenas ao jogo de origem. Todavia, algumas das rotinas da aventura de INPUT podem ser utilizadas, pois são comuns a quase todos os jogos desse tipo.

É o caso de "PEGAR" e "LARGAR": fundamentais para o uso dos objetos, serão inteiramente aproveitadas. A rotina que executa a ordem "LISTAR" também é aproveitável, servindo à maioria dos jogos. Sua aventura ficará bem estranha sem uma função desse tipo.

As demais rotinas dependem das exigências do seu jogo. Para ter uma orientação, observe como programamos as rotinas do nosso jogo. Preste atenção nos locais e nos objetos sobre os quais um determinado verbo pode ter ação.

As rotinas devem ser capazes de verificar se o local e os objetos da ação são apropriados. Devem também estar preparadas para informar os resultados ao jogador, imprimindo mensagens de erro quando ele pedir coisas proibidas.

Uma vez escritas, as rotinas ocuparão o espaço entre as linhas 1390 e 2999, inclusive. O computador selecionará essas rotinas de acordo com o verbo usado pelo jogador.

A MATRIZ G

O Spectrum guarda as descrições dos locais e as linhas das rotinas dos verbos dentro da matriz **G**.

O próximo passo consiste, assim, em colocar os números das linhas iniciais das rotinas em **G**. As linhas 40 a 70 contêm os números. As primeiras três linhas referem-se às descrições dos locais. A última contém os números das linhas iniciais das rotinas dos verbos.

As dimensões da matriz **G** dependem do número de linhas **DATA** e de quantos números a linha **DATA** mais longa contém. O primeiro índice no comando **DIM G (M,N)**, ou seja, **N**, corresponde ao tamanho da linha **DATA** mais longa. O segundo, **M**, à quantidade destas linhas.

Coloque todos os dados nas linhas de 40 a 70 — se eles forem muitos, use números de linha intermediários. Verifique quantos números estão incluídos na linha mais longa. Acrescente às linhas mais curtas tantos zeros quantos forem necessários para que fiquem do mesmo tamanho. Não se esqueça das vírgulas.

Se os números estiverem corretamente colocados em **G**, as linhas 330 a 350 vão selecionar a descrição adequada ao local onde se encontra o jogador. O segundo índice da matriz corresponde à linha onde se encontra o número. Verifique se os índices estão corretos em cada comando **GOTO** de seu novo programa, especialmente se você aumentou o número de linhas **DATA**.

MOVIMENTO

Se o tamanho do mapa de sua aventura for diferente, as linhas 1000 a 1040 deverão ser modificadas, pois tratam do movimento do jogador.

Mais especificamente, as linhas que tratam de movimentos para o norte e para o sul — 1010 e 1030 — precisam ser alteradas se seu mapa não tiver seis locais de largura. A nova largura deve substituir o número 6.

OS OBJETOS

Grandes alterações deverão ser feitas nas linhas 160 a 260, uma vez que seus objetos serão diferentes dos da nossa aventura.

Anote o número de objetos e o comprimento das maiores descrições (tanto das curtas quanto das longas). O número de objetos deve ser o primeiro valor da linha **DATA** 200; será usado como índice nos comandos **DIM** referentes às matrizes de objetos — linha 180 — e para determinar o tamanho de diversos laços **FOR...NEXT**. O segundo índice da matriz **B$** é o comprimento da maior descrição curta e o segundo índice em **SI$**, o da maior descrição longa.

Se usarmos uma linha **DATA** para cada objeto, o programa ficará mais claro. No entanto, se o número de objetos for muito grande, pode-se optar por colocar mais de um por linha. Em todo caso, a ordem deve ser precisa, pois cada grupo de três dados contém elementos de três diferentes matrizes. A ordem é: número do local onde se encontra o objeto, descrição curta e descrição longa. Se o objeto só aparece mais tarde na aventura — após ser encontrado, por exemplo, como o olho — ou se surge ao acaso, como o fiscal da Receita, o número do local deve ser 0.

ROTINA DE AUXÍLIO

Esta é a última rotina a ser alterada. Verifique onde pode ser interessante "AJUDAR" o jogador. Use **IF...THEN** para fazê-lo.

Outras características do jogo original, tais como a rotina que faz surgir de repente o fiscal da Receita — linha 320 —, devem ser alteradas ou apagadas. Podemos também mudar o local onde começa a aventura, na linha 280.

LISTA DE VARIÁVEIS

Para estudar em maior profundidade o nosso programa, use esta lista de variáveis.

G()	matriz contendo os números das linhas de descrição dos locais e de execução de ordens do jogador — execução dos verbos.
R$()	matriz com os verbos e respostas do jogador.
R()	matriz com os números dos verbos.

Elementos correspondentes das duas matrizes acima referem-se ao mesmo verbo.

B()	matriz contendo o local onde cada objeto se encontra.
B$()	matriz com a descrição curta dos objetos.
SI$()	matriz com as descrições longas.

Elementos correspondentes das três matrizes acima referem-se ao mesmo objeto.

NB	número de objetos da aventura.
L	posição do jogador.
LA	sinalizador de estado da lâmpada.
N,S,E,W	saídas de um local.
I$	resposta completa do jogador.
V$	verbo em **I$**.
N$	substantivo em **I$**.
I	contém um número correspondente a um grupo de verbos.
IN	número de objetos carregados pelo jogador.
A$	contém a resposta à pergunta "Quer jogar novamente?"
G	contém o número do objeto deixado de lado pelo jogador — elemento **G** da matriz **B**.

EDIÇÃO NO TRS-80 E NO TRS-COLOR

- COMO TIRAR O MÁXIMO DO COMANDO EDIT
- MODIFICAÇÃO DE CARACTERES
- PROLONGAMENTO DE UMA LINHA
- PROCURA DE UM CARACTERE

Quase todos os micros possuem comandos que facilitam a edição de programas em BASIC, mas o comando EDIT das linhas TRS-80 e TRS-Color é um dos mais sofisticados. Aprenda a utilizá-lo.

Como você já deve ter observado, é muito raro um programa rodar corretamente pela primeira vez. Mensagens de erro, ou problemas que ocorrem durante a execução, tornam imperativo o teste repetido de um programa, até que todos os erros tenham sido eliminados.

Esse processo, chamado de *depuração*, é em geral trabalhoso e demorado, sobretudo se as linhas erradas tiverem que ser inteiramente digitadas. Quase todos os microcomputadores possuem algum tipo de editor de linhas para agilizar o processo de correção de programas. O sistema disponível nos micros das linhas TRS-80 e TRS-Color tem um funcionamento um pouco complicado, mas é um dos mais completos. Neste artigo, você verá como explorar eficientemente os recursos de edição que seu micro lhe oferece.

Suponhamos, por exemplo, que você acabou de descobrir um erro de sintaxe na linha 20 de seu programa. Para acionar o editor de linhas, digite **EDIT 20** e pressione a tecla **<ENTER>**. Nos computadores da linha TRS-Color, aparecerá logo abaixo a linha indicada e, em seguida, outra linha, contendo apenas o número e o cursor de posicionamento. Nos micros da linha TRS-80, o comando **EDIT** não mostrará a linha errada, a não ser que você pressione a tecla **L** uma vez.

```
20 X=RND(1
20 _
```

Como você pode notar, o erro consiste na falta do parêntese à direita.

O computador agora está em *modo de edição*, e você poderá fazer as alterações que quiser na linha apresentada na tela. Caso desista de editá-la, simplesmente pressione a tecla **<ENTER>** uma vez ou, então, a tecla **E** (abreviatura de **END** — fim, em inglês), e o computador voltará ao *modo de comando*.

Uma vez que a linha desejada esteja pronta para ser corrigida, você poderá utilizar uma série de comandos de edição. É neste ponto que os principiantes correm o risco de se confundir, pois para acionar os comandos pressiona-se uma única tecla alfabética (cujo resultado não aparece na tela) ou, então, um número de um ou mais dígitos, seguido da tecla alfabética.

No exemplo anterior, a correção consiste em acrescentar um parêntese ao final da linha. Para isso, pressione a tecla **X**, que aciona o comando de e**X**tensão de linha; o cursor pulará automaticamente para o final da linha 20:

```
20 X=RND(1
20 X=RND(1_
```

Agora você pode digitar o parêntese que falta, pois o computador automaticamente entra em *modo* **I**, ou de *inserção*. Para terminar a linha e sair do modo de edição, pressione **<ENTER>**. Não pressione **E**, pois o computador adicionará esta letra ao final da linha: os comandos de edição só valem quando o modo **I** está desligado.

Vamos ver agora um caso mais complicado. Suponhamos que você queira modificar a linha seguinte, inserindo um novo caractere em algum ponto:

```
50 X=0:Y=0:W=363
```

O valor atribuído à variável **X** deveria ser, por exemplo, 20 e não 0, como está. Para fazer a modificação, digite **EDIT 50** e pressione **<ENTER>**. A linha 50 aparecerá na tela e o cursor logo em seu início. Você precisará deslocá-lo até a posição anterior ao primeiro zero, na linha, para poder inserir o novo caractere (no caso, um **2**). Existem três maneiras de fazer esse deslocamento. A primeira consiste em pressionar repetidamente a barra de espaço: cada vez que ela é pressionada, o cursor avança uma posição, e na tela aparece o caractere da posição seguinte. A segunda consiste em dizer ao editor quantos caracteres deve pular, digitando este número, seguido de uma pressão à barra de espaço. No terceiro método, pressiona-se a tecla **S** (de **S**earch — procurar, em inglês), seguida do caractere para o qual você quer deslocar o cursor.

No exemplo acima, precisamos deslocar o cursor para a posição anterior

ao primeiro zero da linha. Bata duas vezes na barra de espaço ou, então, digite o número **2**, seguido de uma pressão à barra; se preferir, pressione a tecla **S** e, em seguida, a tecla **O**. Seja qual for o método utilizado, o cursor ficará posicionado logo após o sinal de igual. Agora, para inserir o número **2**, você deve entrar em modo de inserção. Para isso, pressione a tecla **I** uma vez; em seguida, acrescente o que quiser à linha. Digite normalmente as adições (pode-se usar, inclusive, o retrocesso). Pressione a tecla **<ENTER>** para sair do modo de edição, uma vez que tenha completado a inserção.

Caso você precise fazer outras correções na linha (por exemplo, acrescentar o número 1 antes do segundo zero, de modo a transformar em 10 o valor de **X**), deverá continuar no modo de edição. Para sair do modo de inserção e voltar ao modo de edição, pressione simultaneamente as teclas **<SHIFT>** e ↑. Com isso, você poderá deslocar o cursor até o segundo zero da linha, usando um dos três métodos explicados acima, digitar **I** para inserir e, finalmente, digitar o número **1**.

Algumas vezes, é necessário substituir caracteres, e não acrescentar outros. Eis aqui um exemplo:

```
70 PRINT "EDIYORA NOVA CULTURA
   L"
```

Chame o comando **EDIT** e posicione o cursor sobre a letra errada (Y). Para substituí-la pelo T, pressione uma vez a letra **C** (de **C**hange — mudar, em inglês) e, em seguida, a letra correta. Se você quiser mudar mais de um caractere, acione novamente a tecla **C** ou, então, o número de caracteres, seguido da tecla **C**. Caso precise substituir, por exemplo, três letras consecutivas, coloque o cursor sobre a primeira e digite **3C**.

Para apagar uma letra, ou uma série de letras, coloque o cursor sobre o caractere que pretende apagar e pressione a letra **D**. Nos computadores da linha TRS-Color, o caractere indicado desaparecerá da tela e os demais serão deslocados para a esquerda, fechando o "buraco". No TRS-80, o caractere apagado será exibido entre dois sinais de exclamação. Para apagar vários caracteres, digite o número de caracteres que quer eliminar, seguido da tecla **D**. Colocando o cursor no primeiro E e pressionando **7D**, por exemplo, você apagará a palavra EDIYORA.

Também é freqüente a necessidade de apagar toda a parte final de uma linha. Suponhamos que você queira apagar o comando PRINT da linha abaixo:

```
10 X=89:Y=76:PRINT "X E Y DEFIN
   IDOS"
```

Entre em modo de edição e coloque o cursor até o ponto da linha a partir do qual se encontram os caracteres que deseja apagar (no exemplo, o segundo sinal de dois pontos). Em seguida, pressione a tecla **H** (abreviatura de **H**ack— podar, em inglês) e o resto da linha desaparecerá. Este comando também entra no modo de inserção, permitindo que você acrescente algo à linha ou que abandone o modo de edição, pressionando (ENTER).

Você pode também apagar partes de uma linha utilizando o comando **K** (de **K**ill — matar, em inglês). Ele elimina caracteres sucessivamente, até que seja encontrada a primeira ocorrência do caractere especificado. Se você digitar, por exemplo, **KD**, eliminará todos os caracteres entre a posição do cursor e a primeira ocorrência da letra D, na linha. Caso digite **3KD**, apagará tudo até a terceira ocorrência de D.

Depois de fazer algumas modificações em uma linha, liste-a, ainda dentro do modo de edição, para ver como ficou. Para isso, digite a tecla **L**.

Se você cometeu algum erro e deseja cancelar todas as modificações feitas, digite a tecla **A** (**A**gain — novamente, em inglês); o cursor retornará ao seu início. O comando **Q** (**Q**uit — abandonar, em inglês) faz a mesma coisa, mas sai do modo de edição.

Para se exercitar um pouco, digite o programa abaixo:

```
10 CLS
20 PRINT
30 PRINT "NU,MERO","CUBO"
40 FOR A=1 TO 13 STEP 2
50 PRINT A,*A*A
60 NEXT
70 PRINT "ESTE E' O FIM"
```

Ele contém alguns erros; seu aspecto final deverá ser o seguinte:

```
10 CLS
20 PRINT "TESTE"
30 PRINT "NUMERO","CUBO"
40 FOR A=1 TO 13
50 PRINT A,A*A*A
60 NEXT
70 PRINT "FIM"
```

Para alterar o programa, procure usar todos os comandos de **EDIT** que aprendeu. Eis aqui a notação abreviada das modificações que poderá tentar:

```
EDIT 20 X"TESTE" [ENTER]
EDIT 30 S,D[ENTER]
EDIT 40 4[ESPAÇO]H[ENTER]
EDIT 50 S,IA[ENTER]
EDIT 70 SEKOD[ENTER]
```

Sumário dos comandos de EDIT

L	Lista a linha
C caractere	Muda caractere
nC caracteres	Muda os próximos n caracteres
I	Insere caracteres
D	Apaga um caractere
nD	Apaga n caracteres
H	Apaga o restante da linha e entra em modo de inserção
X	Entra em modo de inserção ao final da linha
S caractere	Procura a primeira ocorrência do caractere
nS caractere	Procura a n-ésima ocorrência do caractere
K caractere	Apaga tudo até a primeira ocorrência do caractere
nK caractere	Apaga tudo até a n-ésima ocorrência do caractere
<ESPAÇO>	Avança o cursor uma posição
n<ESPAÇO>	Avança o cursor n posições
←	Recua o cursor de uma posição
n ←	Recua o cursor de n posições
<SHIFT> ↑	Sai do modo de inserção e volta ao modo de edição
<ENTER>	Sai do modo de edição e inserção
E	Sai do editor
A	Retorna ao início da edição
Q	Retorna ao início da edição e sai do editor.

ERRATA

VOLUME I *Continuação I*

Pág. 6 - 2ª col. - quadro
Corrigir a 2ª linha de códigos numéricos. *Onde se lê:*
151 131 190 176 280
Leia-se:
151 131 191 176 280

Pág. 24 - 2ª col. - programa
Para funcionar no TRS-80, substitua a linha 8 por:

```
8 CLS
```

Pág. 30 - 1ª col. - 1º programa
Corrija a linha 40:

```
40  IF K$="F" THEN  M=374  ELSE GOTO 30
```

Pág. 31 - 2ª col. - 3º parágrafo
Onde se lê:
O programa para o TRS-80 funciona de modo idêntico ao programa para o TRS-Color.
Adicione:
...com exceção do fato de que as teclas a serem pressionadas são L e R, respectivamente, para comandar movimentos à esquerda e à direita.

Pág. 33 - 3ª col. - programa
Corrija a linha 150:

```
150 IF PO>1022 OR PO<0 THEN PO=
LP:GOTO 50
```

Pág. 62 - 1ª col. - programa
O logotipo não se aplica a este programa

Pág. 63 - 2ª col. - penúltimo parágrafo
Onde se lê:
...para o TRS-80 e TRS-Color.
Leia-se:
...para o TRS-Color.

Pág. 224 - 1ª e 2ª col. - programa
Adicione:
O programa indicado funciona apenas no TRS-Color. Para executá-lo no TRS-80 faça as seguintes modificações: multiplique por dois todos os números depois dos comandos PRINT@, nas linhas 10, 20, 30, 60, 90 e 110; suprima as linhas 15 e 120 e adicione a linha:

```
95 TIMER=TIMER+0.1
```

Pág. 270 a 275
Todos os programas indicados para o TRS-Color também funcionam no TRS-80.

Pág. 19 - 2ª col. - programa
Corrija a linha:

```
430 RESTORE:FOR J=0 TO N-1:READ
A$:PRINT#-P,STRING$(LL-ML," ");
SP$;:PRINT#-P,MID(A$,2);
CHR$(13);
```

Pág. 31 - 2ª col. - 3º parágrafo
Onde se lê:
A linha 50 investiga se "L" foi pressionada.....A linha 70 confere se "R" foi pressionada...
Leia-se:
A linha 50 investiga se "E" foi pressionada.....A linha 70 confere se "D" foi pressionada...

Pág. 45 - 1ª col. - 5º parágrafo
Onde se lê:
...apenas os das linhas TRS-80 e MSX...
Leia-se:
...apenas os das linhas TRS-80, TRS-Color e MSX...

Pág. 65 - 1ª col. - programa
Corrija a linha 420 para:

```
420 LET K$=INKEY$:IF K$="" THEN
GOTO 420
```

Pág. 90 - 3ª col. - 4º parágrafo
Onde se lê:
...limitar a memória em §H3680...que se reserve memória só até §H3680....só poderá usar o **CLEAR** até §H1E80.
Leia-se:
...limitar a memória em &H3680...que se reserve memória só até &H3680....só poderá usar o **CLEAR** até &H1E80.

Pág. 219 - 2ª col. - 3º parágrafo
Onde se lê:
PSHB armazena temporariamente...
Leia-se:
PSHS armazena temporariamente...

Pág. 10 - 3ª col. - 3º parágrafo
As linhas 20 HTAB 18... e 20 LOCATE...
estão trocadas de lugar.

Pág. 16 - 1ª col. - programa
Substitua a linha 140 por:

```
140 GOTO 80
```

Pág. 30 - 1ª col. - programa
Corrija a linha 60:

```
60 LOCATE 18,M:PRINT "^"
```

Pág. 32 - 2ª col. - 2º parágrafo
Onde se lê:
O jogo termina quando a tecla CLEAR (código 26)...
Leia-se:
O jogo termina quando a tecla ESCAPE (código 27)...

Pág. 33 - 1ª col. - 1º parágrafo
Adicione:
Nos micros da linha MSX, as teclas a serem utilizadas para deslocamento da base e disparo são as flechas do cursor e da barra de espaços, repectivamente (correspondentes aos códigos 28, 29 e 32)

Pág. 58 - 3ª col. - programa
Corrija a linha 330:

```
330 LOCATE 2+X*3,18:PRINT USING
"#";N;
```

Pág. 73 - 1ª col. - programa
Corrija as linhas:

```
20 CLEAR 5000:R$="P RMCAI"
30 CLS:LOCATE 5,1:PRINT"M E N U
 P R I N C I P A L"
```

Pág. 74 - 2ª col. - programa
Corrija a linha:

```
10020 IN$=INKEY$:IF IN$="" THEN
10020
```

Pág. 84 - 2ª col. - programa
Corrija a linha:

```
3020 IN$=INKEY$:IF IN$=""  THEN
3020
```

Pág. 85 - 1ª col. - programa
A linha 5070 tem esta redação:

```
5070  IFD>NR AND G=1 THEN  G=0:
CH=-1 ELSE IF D>NR THEN 5230
```

Pág. 89 - 3ª col. - 1º parágrafo
Onde se lê:
SPRITE(0) A$
Leia-se:
SPRITE(0)=A$

Pág. 93 - 1ª col. - programa
Corrija a linha:

```
250 POKE SA,VAL("&H0"+S$)
```

Pág. 107 - 2ª col. - penúltimo parágrafo
Pág. 119 - 3ª col. - 2º parágrafo
É possível desenhar letras e gráficos na mesma tela do MSX, porém usamos o comando DRAW neste programa, de modo a compatibilizá-lo com as versões para outros computadores.

Pág. 120 - 3ª col. - último parágrafo
Onde se lê:
As cores disponíveis no PMODE 3,1 são:
verde (1), amarelo (2), azul (3) e vermelho (4).
Leia-se:
As cores disponíveis para todos os **SCREEN** no MSX são: (1) preto, (2) verde, (3) verde claro e (4) azul escuro.

Continua no próximo número

NO PRÓXIMO NÚMERO

APLICAÇÕES
Com um programa especial, seu computador poderá ajudá-lo a executar belíssimos trabalhos gráficos. Experimente-o.

CÓDIGO DE MÁQUINA
Um Assembler para o MSX: deixe que o computador faça por você a tradução dos mnemônicos Assembly para hexadecimal.

CÓDIGO DE MÁQUINA
Mesmo sem conhecer código de máquina, você pode animar seus jogos com figuras simples. Veja como é fácil.